Problema dei tubi

Descrizione del problema

Il progetto consiste nella risoluzione di un problema di ottimizzazione per quanto riguarda una fabbrica di cioccolato.

La suddetta fabbrica presenta una tanica di cioccolato (*radice*) dalla quale partono diverse tubature che portano il cioccolato alle stazioni di produzione; queste tubature sono interconnesse da dei giunti (*nodi*) che periodicamente devono essere manutenuti, e questo richiede del tempo.

Sono però disponibili dei giunti (*nodi*) di ricambio che azzerano il tempo di manutenzione.

Si deve quindi trovare quali giunti nella diramazione di tubi è ottimo cambiare per ottenere il minor tempo possibile per l'inizio della produzione.

Input

La prima riga contiene due numeri interi N, C; il primo rappresenta il numero totale di giunti presenti nel sistema, mentre il secondo rappresenta il numero di ricambi disponibili. Le righe successive sono composte dal parametro t_n (*tempo di manutenzione*) e p_n (*nodo genitore*).

```
N C
t0 p0
t1 p1
 ...
ti pi
```

Vincoli

- Intero $1 \leq N \leq 10000$
- Intero $0 \leq C \leq 100$
- Intero $0 \leq T_i \leq 10^4$
- Un solo valore $P_i = -1$ mentre per tutti gli altri $0 \leq P_i < N$
- Esiste sempre una sequenza che da n_i conduce alla tanica principale
- Si è grado di effettuare le operazioni di manutenzione in parallelo

Output

Un singolo valore intero che rappresenta il minimo tempo entro il quale l'ultima stazione può iniziare la produzione.

Requisiti

Viene richiesto di elaborare una soluzione che trovi la soluzione ottima al problema sfruttando il paradigma della programmazione dinamica.

Soluzione

Analizzando la richesta del problema, abbiamo intuito che l'algoritmo richiesto dovesse risolvere un problema di ottimizzazione su un albero radicato, dove ogni nodo ha un valore associato (il tempo di manunenzione) e dove si dispone di "pezzi di ricambio", con i quali si può azzerare il valore associato.

L'obbiettivo è quello di minimizzare il massimo valore nei sottoalberi risultanti, distribuendo ricambi in modo ottimale tra i nodi.

È quindi palese la presenza di una struttura arborescente, rappresentante in toto il problema, della quale vanno esplorati i nodi, per poi calcolarne il massimo valore.

Questo problema è stato dapprima affrontato consultando il materiale del corso, nel quale abbiamo trovato differenti opzioni per la visita ricorsiva dei nodi, partendo da una radice; di questi abbiamo subito scelto di approcciarci al problema sulla base del prototipo del `DFS`.

Riassumendo, l'algoritmo percorre l'albero, dalla radice alle foglie, dando priorità alla profondità, calcolando tutte le combinazioni di ricambi nel sottoalbero del nodo corrente, per poi minimizzare il massimo tra il risultato del sottoalbero e quello del nodo corrente.

L'algoritmo è stato implementato quindi ricorsivamente, il quale parte dalla radice e visita i nodi seguendo una trasversale *post-ordine*, quindi vengono visitati dapprima i figli, poi il nodo corrente. I risultati calcolati per i figli vengono combinati per calcolare il risultato del nodo corrente.

Pseudocodice

Algorithm 1 Minimize Time

```
procedure MinimizeTime(n)
    if child_count[n] = 0 then                                  ▷ Caso base
        mem[n][0] ← T[n]
        for c ← 1 to C do
            mem[n][c] ← 0
        end for
        return
    end if
    for each child v of n in adj[n] do                          ▷ Caso generale
        MinimizeTime(v)
        Initialize cur[0 . . . C] ← ∞
        for c ← 0 to C do
            for x ← 0 to c do
                cur[c] ← min(cur[c], max(mem[v][x], mem[node][c − x]))
            end for
        end for
        mem[n][...] ← cur[...]
    end for
    Initialize cur[0 . . . C]
    cur[0] ← T[n] + mem[n][0]
    for c ← 1 to C do                                           ▷ Wrap-up
        cur[c] ← min(T[n] + mem[n][c], mem[n][c − 1])
    end for
    mem[n][...] ← cur[...]
end procedure
```

Come si può carpire dallo pseudocodice, l'algoritmo si compone di tre passaggi:

1. Caso base Il caso base rappresenta l'evenienza nel quale l'algoritmo si trova su un *nodo foglia*, quindi un nodo dove non sono presenti figli.

In questo caso l'algoritmo ha due cammini che può seguire:

- se il numero di ricambi residui c è 0, il valore massimo è semplicemente il valore del nodo stesso;
- se invece il numero di ricambi residui è c strettamente maggiore di 0, il massimo è 0, dato che non ci sono figli su cui utilizzarli.

2. Caso generale Per un nodo con u figli vengono dapprima calcolate le distribuzioni di ricambi sul sottoalbero:

- Vengono iterati tutti i figli v di n e si combinano le soluzioni calcolate per i sottalberi radicati in v.
- Viene tentata ogni possibile distribuzione x dei c ricambi dove:
  - x ricambi vengono applicati al sottoalbero del figlio v;
  - $c - x$ ricambi vengono applicati ai rimanenti figli e a n.

La formula utilizzata risulta quindi essere:

$$cur(c) = \min(cur(c), \max(S_v(x), S_u(c - x)))$$

dove $cur(c)$ rappresenta il valore attuale calcolato per c ricambi distribuite nel sottoalbero, $S_v(x)$ la soluzione ottimale per x ricambi applicati al figlio v e $S_n(c - x)$ è la soluzione ottimale dei rimanenti $c - x$ ricambi applicati al nodo n o ad altri suoi figli.

Questa minimizza il risultato massimo del sottoalbero del figlio v e quello del nodo corrente.

3. "Wrap-up" Viene infine considerato il miglioramento direttamente sul nodo corrente n, decisione che viene salvata nella memoria di programmazione dinamica:

$$mem(n, c) = \min(T_n + mem(n, c), mem(n, c - 1))$$

Statistiche di tempo

La soluzione da noi implementata risulta abbastanza veloce da essere giudicata soddisfacente, a nostro avviso; abbiamo constatato una i seguenti valori statistici (considerati per 10 esecuzioni della stessa istanza):

Dato	Valore
Media $\overline{x}$	175.3 µs
Mediana $\widetilde{x}$	172.5 µs
Range	200.0 µs
Minimo	117.0 µs
Massimo	317.0 µs

Questi dati si riferiscono all'insieme di task di numerosità n di 30, per un totale tempo di esecuzione di 5.259 ms.

La complessità temporale risulta quindi essere $O(N + C^2)$, dove N è la grandezza dell'input e C è il numero di ricambi disponibili.

Statistiche di spazio

Sviluppi futuri

La complessità spaziale dell'algoritmo può essere migliorata notevoltmente, passando da una gestione della memoria statica, presente nella versione attuale dell'algoritmo, ad una dinamica, allocando solamente lo spazio strettamente necessario per ogni task.

Conclusione