Algoritmi e Strutture Dati
Università degli Studi di Trieste **Homework 4** 07/01/2025

Esempi di domande di sbarramento

Rispondere correttamente alle seguenti domande:

1. Qual è la complessità di quicksort nel caso peggiore?
   (a) $O(n \log n)$
   (b) $\Omega(n^2 \log n)$
   (c) $O(n^2)$
   (d) $O(n)$
2. Qual è la complessità della ricerca in un albero binario di ricerca con n nodi nel caso peggiore?
   (a) $O(\log n)$
   (b) $O(n)$
   (c) $\Omega(n \log n)$
   (d) $\Omega(n^2)$
3. Qual è la complessità della visita in profondità nel caso peggiore?
   (a) $O(V)$
   (b) $\Theta(VE)$
   (c) $O(E)$
   (d) $O(V + E)$
4. Hai un grafo orientato connesso con pesi positivi. Devi trovare il cammino minimo da un nodo s a un nodo t e poi continuare fino a un altro nodo z. Quale approccio useresti?
   (a) Esegui una sola volta Dijkstra da s
   (b) Esegui Dijkstra due volte: da s a t, poi da t a z
   (c) Usa Bellman-Ford
   (d) Ordina topologicamente il grafo e usa un approccio diretto
5. Hai un grafo non orientato con n nodi e m archi. Vuoi determinare se il grafo è bipartito. Quale algoritmo è più efficiente?
   (a) BFS con colorazione alternata dei nodi
   (b) Kruskal
   (c) Bellman-Ford

(d) DFS e verifica delle parità dei cicli

6. Devi determinare se un grafo diretto contiene un ciclo. Quale approccio è più efficiente?

    (a) Usa un algoritmo di Prim

    (b) Usa Dijkstra

    (c) Esegui una visita in profondità (DFS) e controlla i nodi nel percorso attivo

    (d) Usa una matrice di adiacenza per calcolare il numero di archi entranti e uscenti

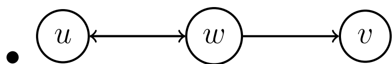
Grafi

Esercizio 1

Dare due controesempi per le seguenti affermazioni, in cui $x.d$ e $x.f$ denotano rispettivamente il tempo di inizio e fine visita del nodo x:

- Se un grafo diretto G contiene un cammino da u a v e se $u.d < v.d$ in un'esecuzione di $DFS(G)$, allora v è un discendente di u in uno degli alberi della foresta DFS.
- Se un grafo diretto G contiene un cammino da u a v, allora per ogni esecuzione di $DFS(G)$ si ha $v.d \leq u.f$.

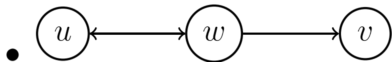
Soluzione

- 

  Nel grafo precedente, un'esecuzione di DFS che parte dal nodo w e processa u prima di v produce

  $$w.d \leq u.d \leq u.f \leq v.d \leq v.f \leq w.f$$

  Dunque c'è un cammino da u a v e $u.d < v.d$ ma nell'albero DFS u e v sono entrambi figli di w, dunque v non è discendente di u.

- 

  Nel grafo precedente, un'esecuzione di DFS che parte da w e processa u prima di v produce

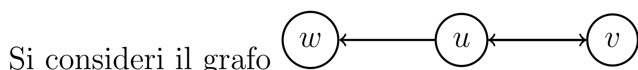
  $$w.d < u.d < u.f < v.d < v.f < w.f$$

  Dunque, anche se c'è un cammino da u a v si ha $v.d > u.f$

Esercizio 2

A tutorato Francesca sostiene che nell'algoritmo di Tarjan per il calcolo delle componenti fortemente connesse, sostituendo G nella seconda esecuzione di DFS e poi esplorando i nodi in ordine crescente di tempo di fine visita, l'algoritmo sarebbe più efficiente ma comunque corretto. Mostrare con un controesempio che l'algoritmo non è corretto. Descrivere come l'algoritmo di Tarjan (originale) agisce sul controesempio.

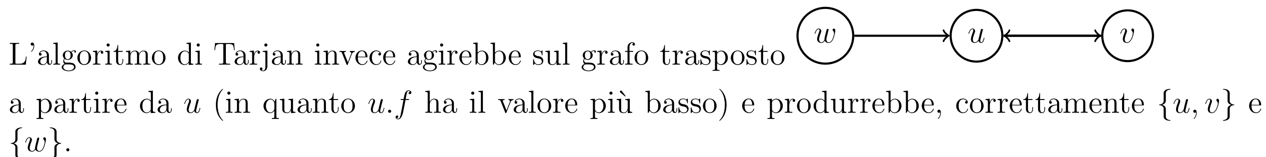
Soluzione

Si consideri il grafo 

Nel grafo precedente, un'esecuzione di DFS che parte da u e processa v prima di w produce:

$$u.d < v.d < v.f < w.d < w.f < u.f$$

Nella versione modificata dell'algoritmo di Tarjan, DFS verrebbe eseguito su v (in quanto $v.f$ ha il valore più basso) dal momento che da v entrambi gli altri nodi sono raggiungibili, l'algoritmo restituirebbe un'unica componente connessa $\{u, v, w\}$.

L'algoritmo di Tarjan invece agirebbe sul grafo trasposto 

a partire da u (in quanto $u.f$ ha il valore più basso) e produrrebbe, correttamente $\{u, v\}$ e $\{w\}$.

Programmazione Dinamica

Esercizio 3

Dovete pianificare una dieta giornaliera. Avete a disposizione n alimenti che forniscono rispettivamente $c[1], \ldots, c[n]$ calorie. In base alle vostre preferenze, stimate che ogni alimento richieda rispettivamente $t[1], \ldots, t[n]$ minuti per essere preparato. Purtroppo il tempo a vostra disposizione per cucinare è T minuti, che potrebbe essere inferiore alla somma dei tempi necessari a preparare tutti gli alimenti. Le calorie e i tempi sono numeri interi strettamente positivi.

- Scrivere un algoritmo efficiente che, dati i vettori $c[1..n]$, $t[1..n]$ e il valore di T, restituisce il massimo numero di calorie che potete ottenere selezionando un opportuno sottoinsieme dei n alimenti entro il tempo massimo di T minuti.
- Calcolare il costo computazionale dell'algoritmo proposto.

Soluzione
Definiamo $C[i, j]$ come il massimo numero di calorie che si possono ottenere selezionando un sottoinsieme degli alimenti $\{1, \ldots, i\}$ avendo a disposizione un tempo massimo di j minuti. Questo significa che i due indici che definiscono C avranno valore $i \in \{1, \ldots, n\}$ e $j \in \{1, \ldots, T\}$.
Per definire la tabella di programmazione dinamica occorre capire come definire $C[i, j]$. Ci sono due possibilità:

- Se il tempo necessario a preparare l'alimento i-esimo eccede il tempo a disposizione ($j < t[i]$), allora non possiamo includerlo, e il massimo numero di calorie è quello che si poteva ottenere selezionando un sottoinsieme degli alimenti $\{1, \ldots, i-1\}$, cioè $C[i-1, j]$.
- Se invece abbiamo il tempo per preparare l'alimento, possiamo scegliere se includerlo o meno. Includendolo, il numero di calorie aumenta di $c[i]$, e il nuovo tempo a disposizione diventa $j - t[i]$. Pertanto, il massimo numero di calorie sarà $\max\{C[i-1, j], C[i-1, j-t[i]] + c[i]\}$.

Queste informazioni si possono inglobare nella seguente definizione ricorsiva della tabella:

$$C[i, j] = \begin{cases} \max\{C[i-1, j], C[i-1, j-t[i]] + c[i]\} & \text{se } j \geq t[i] \\ C[i-1, j] & \text{altrimenti} \end{cases}$$

Per completare la definizione dell'algoritmo, occorre inizializzare la tabella C nel seguente modo:

$$C[1, j] = \begin{cases} c[1] & \text{se } j \geq t[1] \\ 0 & \text{altrimenti} \end{cases}$$

Algorithm 1 MaxCalories($c[1 \dots n]$ array **int**; $t[1 \dots n]$ array **int**; T **int**)

```
1: Inizializzare C[1...n; 0...T] di int
2: for j = 0,...,T do
3:     if j ≥ t[1] then
4:         C[1, j] = c[1]
5:     else
6:         C[1, j] = 0
7:     end if
8: end for
9: for i = 2,...,n do
10:    for j = 0,...,T do
11:        if j ≥ t[i] and C[i − 1, j] < C[i − 1, j − t[i]] + c[i] then
12:            C[i, j] = C[i − 1, j − t[i]] + c[i]
13:        else
14:            C[i, j] = C[i − 1, j]
15:        end if
16:    end for
17: end for
18: return C[n, T]
```

Possiamo riassumere il ragionamento nel seguente algoritmo:

Costo computazionale: L'algoritmo ha complessità temporale $O(n \cdot T)$ e complessità spaziale $O(n \cdot T)$ poiché la tabella C ha dimensioni $n \times T$.

Esercizio 4

Un bartender ha a disposizione n ingredienti liquidi i cui volumi sono rispettivamente $v[1], v[2], \ldots, v[n]$, dove $v[i]$ sono interi positivi ed è possibile che più ingredienti abbiano lo stesso volume.
Studiamo il problema di decidere se sia o meno possibile preparare un cocktail con un volume totale esattamente pari a V utilizzando un opportuno sottoinsieme degli n ingredienti a disposizione dove, nuovamente, V è un intero positivo.

- Descrivere lo **pseudo-codice** di un algoritmo efficiente per decidere se il problema ammette una soluzione. L'algoritmo, quindi, dovrà tornare un valore di verità (*true* se si può preparare il cocktail, *false* altrimenti). Attenzione: l'algoritmo non richiede di minimizzare il numero di ingredienti utilizzati, ma chiede semplicemente se sia possibile raggiungere il volume corretto.
- Calcolare il costo computazionale dell'algoritmo proposto.

Soluzione
Definiamo la matrice booleana $M[1, \ldots, n; 0, \ldots, V]$ dove $M[i, j] = \textit{true}$ se è possibile preparare un cocktail con volume esattamente pari a j utilizzando un sottoinsieme degli ingredienti $\{1, \ldots, i\}$.
La tabella si definisce in modo ricorsivo nel seguente modo:

- Se il volume del i-esimo ingrediente è maggiore di j, l'ingrediente non può essere usato, quindi $M[i, j] = M[i-1, j]$.
- Se invece il volume del i-esimo ingrediente è minore o uguale a j, possiamo decidere se includerlo o meno. Se decidiamo di includerlo, allora il nuovo volume residuo sarà $j - v[i]$. Pertanto:
$$M[i, j] = M[i-1, j] \vee M[i-1, j - v[i]]$$

Per inizializzare correttamente la tabella, abbiamo:

$$M[1, j] = \begin{cases} \textit{true} & \text{se } j = 0 \text{ o } j = v[1] \\ \textit{false} & \text{altrimenti} \end{cases}$$

Lo pseudocodice segue direttamente da queste definizioni, vedi es. precedente.

Costo computazionale: L'algoritmo ha complessità temporale $O(n \cdot V)$ e complessità spaziale $O(n \cdot V)$, poiché la matrice M ha dimensioni $n \times V$.