Algoritmi e Strutture Dati

Lezione 35

15 dicembre 2025

Cognome.....

Nome.....

Matricola.....

Algoritmi e Strutture Dati

Prova scritta del 2 febbraio 2024

TEMPO DISPONIBILE: 2 ore

Le risposte agli esercizi 1 e 2 devono essere scritte negli appositi riquadri su questo foglio (risposte scritte su altri fogli non saranno considerate). La soluzione dell'esercizio 3 va scritta su uno dei fogli di protocollo forniti. Le brutte copie NON devono essere consegnate. Ricordatevi di scrivere cognome e nome su TUTTI i fogli (inclusi quelli di brutta).

1. Considerate la sequenza di numeri `37 22 49 52 46 60 35`.

(a) Disegnate l'albero AVL che si ottiene inserendo i numeri della sequenza, uno dopo l'altro, nell'ordine indicato, a partire da un albero inizialmente vuoto, ribilanciando quando necessario, in modo da ottenere un albero AVL dopo ogni inserimento.	(b) Se l'albero ottenuto al punto (a) è *perfettamente bilanciato* allora scrivete la definizione di albero perfettamente bilanciato, altrimenti disegnate un albero di ricerca perfettamente bilanciato contenente le stesse chiavi.

2. Considerate la sequente sequenza di numeri, memorizzata in un array: `35 50 37 22 49 52 46 60 55`.

(a) Supponete di ordinare la sequenza *in modo crescente* mediante l'algoritmo `quickSort`, scegliendo come perno il primo elemento ed utilizzando, per effettuare la partizione, la strategia presentata a lezione.
Indicate il contenuto dell'array dopo avere effettuato la partizione, prima delle chiamate ricorsive di `quickSort`.

(b) Supponete di ordinare la sequenza *in modo crescente* mediante l'algoritmo `bubbleSort`.
Indicate il contenuto dell'array al termine dell'iterazione del ciclo principale nella quale la chiave `50` viene collocata nella sua posizione definitiva.

(c) Considerate le sequenze formate dai primi 4 e dagli ultimi 5 elementi della sequenza data.
Supponete di fonderle con l'algoritmo di `merge` (rispetto all'*ordine crescente*) senza averle ordinate.
Indicate la sequenza che si ottiene.

(d) Supponete di ordinare la sequenza *in modo decrescente* mediante l'algoritmo `mergeSort`.
Indicate il contenuto dell'array dopo l'esecuzione delle due chiamate ricorsive, prima del `merge` finale.

3. Una banda di ladri ha deciso di trascorrere qualche giorno di vacanza in una zona di montagna quasi disabitata, andando ad occupare una delle molte baite presenti.[1] Per ridurre la possibilià di incontrare estranei, il capo della banda decide che dovrà essere individuata la baita che richiede più tempo per essere raggiunta *partendo dalle altre baite*.

Le baite sono collegate tra loro da sentieri che possono essere percorsi solo a piedi. Si conosce il tempo di percorrenza di ciascun sentiero che collega direttamente tra loro due baite. Tuttavia, a causa delle caratteristiche del terreno e delle presenza di molti ostacoli, il tempo di percorrenza di un sentiero in una direzione può essere diverso dal tempo di percorrenza nella direzione opposta. Per le stesse ragioni, inoltre, alcuni sentieri possono essere percorsi solo in una direzione. Comunque, partendo da ogni baita è sempre possibile raggiungere ogni altra baita, seguendo i sentieri opportuni.

Per risolvere questo problema, i ladri devono progettare un algoritmo che, ricevendo in ingresso le informazioni relative ai sentieri che collegano le baite, stabilisca in quale baita B devono trascorrere le vacanze, in modo che la media dei tempi per raggiungere la baita B *dalle altre baite* risulti *massima*. L'algoritmo dovrà prima di tutto calcolare, per ogni baita, la media dei tempi per raggiungerla *dalle altre baite* e, successivamente, scegliere la baita con tempo medio massimo.

Cosa si richiede

Risolvete i seguenti punti ***nell'ordine indicato.*** Prima di iniziare leggete anche le note scritte alla fine.

(a) Spiegate come il problema possa essere descritto e formalizzato in termini di grafi.

(b) Indicate quale degli algoritmi presentati a lezione può essere adattato per risolvere questo problema e spiegate sinteticamente le modifiche da effettuare a tale scopo. Scrivete poi lo pseudocodice dell'algoritmo modificato. L'algoritmo deve ricevere in ingresso la matrice T specificata nelle note e deve fornire in uscita il nome della baita scelta e la media dei tempi per raggiungerla dalle altre.

(c) Fornite una stima del tempo di calcolo impiegato dall'algoritmo trovato al punto precedente in funzione del numero n di baite, *giustificando adeguatamente la risposta*.

(d) Si deve produrre un elenco che indichi, per ogni coppia di baite x e y, quanto tempo serve per andare da x a y. L'elenco deve essere in ordine *non crescente* di tempo. Indicate come si può modificare ed estendere a questo scopo l'algoritmo ottenuto al punto (b), eventualmente introducendo strutture dati aggiuntive. Potete richiamare un algoritmo di ordinamento noto (indicate quale, senza descrivere la stategia).

(e) Fornite una stima del tempo di calcolo della parte descritta al punto (d), ricordandovi che n è il numero di baite.

Note

- I nomi delle baite sono gli interi da 1 a n.
- L'algoritmo riceve in input una matrice T con n righe e n colonne tale che per $i, j = 1, \ldots, n$, l'elemento di riga i e colonna j è:

$$T_{i,j} = \begin{cases} 0 & \text{se } i = j \\ \text{tempo per percorrere il sentiero dalla baita } i \text{ alla baita } j & \text{se c'è un sentiero che partendo dalla baita } i \text{ arriva direttamente alla baita } j, \text{ senza passare per altre baite} \\ \infty & \text{altrimenti.} \end{cases}$$

- Le risposte devono essere adeguatamente giustificate.
- Non cambiate i nomi stabiliti nel testo dell'esercizio, in particolare T.
- Prestate attenzione alle direzioni dei sentieri e al significato di righe e colonne nelle matrici!

Esempio

Matrice in ingresso:

$$T = \begin{bmatrix} 0 & 5 & \infty & \infty \\ \infty & 0 & 7 & 4 \\ \infty & 1 & 0 & \infty \\ 2 & \infty & 2 & 0 \end{bmatrix}$$

Calcolando per tutte le coppie di baite il tempo che serve per andare da una baita all'altra si ottengono i valori nella seguente matrice:

$$\begin{bmatrix} 0 & 5 & 11 & 9 \\ 6 & 0 & 6 & 4 \\ 7 & 1 & 0 & 5 \\ 2 & 3 & 2 & 0 \end{bmatrix}$$

La baita richiesta è 3, il tempo medio *dalle* altre baite *a* 3 è 6.33.

Elenco ordinato tempi (punto (d))

da	a	tempo
1	3	11
1	4	9
3	1	7
2	1	6
2	3	6
1	2	5
3	4	5
2	4	4
4	2	3
4	1	2
4	3	2
3	2	1

[1] *Baita:* Piccola costruzione di montagna utilizzata in alcuni periodi dell'anno come abitazione, deposito di bestiame, raccolto, materiali.

(a) Spiegate come il problema possa essere descritto e formalizzato in termini di grafi.

Grafo orientato fortemente connesso

V vertici : rappresentano le basi

E archi : " i sentieri

pesi archi : " i tempi di percorrenza

(b) Indicate quale degli algoritmi presentati a lezione può essere adattato per risolvere questo problema e spiegate sinteticamente le modifiche da effettuare a tale scopo. Scrivete poi lo pseudocodice dell'algoritmo modificato. L'algoritmo deve ricevere in ingresso la matrice T specificata nelle note e deve fornire in uscita il nome della baita scelta e la media dei tempi per raggiungerla dalle altre.

(c) Fornite una stima del tempo di calcolo impiegato dall'algoritmo trovato al punto precedente in funzione del numero n di baite, *giustificando adeguatamente la risposta.*

Algoritmo di Floyd & Warshall

```
ALGORITMO sceglibaita (Matrice T[1..n,1..n]) → (Intero, Float)
    FOR k←1 TO n DO
        FOR i←1 TO n DO
            FOR j←1 TO n DO
                IF T[i,k] + T[k,j] < T[i,j]
                    THEN T[i,j] ← T[i,k] + T[k,j]

    mediaMax ← 0
    baitaMax ← 0
    FOR j ← 1 TO n DO
        somma ← 0
        FOR i ← 1 TO n DO
            somma ← somma + T[i,j]
        media ← somma/(n-1)
        IF media > mediaMax THEN
            mediaMax ← media
            baitaMax ← j
    RETURN (baitaMax, mediaMax)
```

(d) Si deve produrre un elenco che indichi, per ogni coppia di baite x e y, quanto tempo serve per andare da x a y. L'elenco deve essere in ordine *non crescente* di tempo. Indicate come si può modificare ed estendere a questo scopo l'algoritmo ottenuto al punto (b), eventualmente introducendo strutture dati aggiuntive. Potete richiamare un algoritmo di ordinamento noto (indicate quale, senza descrivere la stategia).

(e) Fornite una stima del tempo di calcolo della parte descritta al punto (d), ricordandovi che n è il numero di baite.

- Scorro la matrice T ottenuta per ogni elemento (salvo i=j) creo un tupla $< i, j, T[i,j] >$ che inserisco in array A di grandezza $n^2 - n$
- Ordino A in base al terzo elemento delle tupla (Temp) in modo non crescente con HeapSort

Tempi: - Scorrere matrice e riempire array $\Theta(n^2)$

- ordinamento $\Theta(n^2 \lg n)$

$\Theta(n^2 \lg n)$

Algoritmi e Strutture Dati

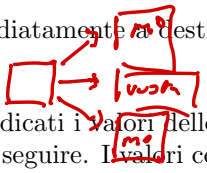
Prova scritta del 16 settembre 2022

TEMPO DISPONIBILE: 1 ora e 30 minuti

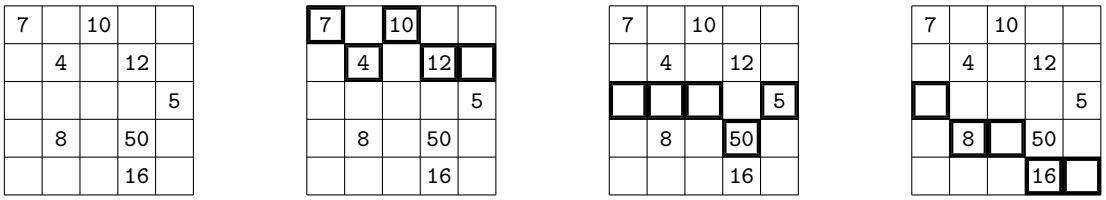
Scrivete il vostro nome e cognome su tutti i fogli. Le copie di brutta e questo foglio non vanno consegnati.

È data una griglia quadrata costituita da n righe e n colonne alle cui intersezioni si trovano dei quadrati. In alcuni di questi è collocata una moneta di valore intero positivo. Quando il giocatore raggiunge un quadrato contenente una moneta se ne impossessa. *Obiettivo* del giocatore è massimizzare il valore complessivo delle monete delle quali entra in possesso, muovendosi lungo un cammino che inizia da un qualunque quadrato della colonna più a sinistra, termina in un quadrato della colonna più a destra e rispetta le seguenti regole:

- da un quadrato il giocatore si può muovere nel quadrato immediatamente a destra sulla stessa riga, se quest'ultimo è vuoto;
- da un quadrato il giocatore si può muovere nel quadrato immediatamente a destra sulla riga immediatamente superiore (se esiste), se quest'ultimo contiene una moneta;
- da un quadrato il giocatore si può muovere nel quadrato immediatamente a destra sulla riga immediatamente inferiore (se esiste), se quest'ultimo contiene una moneta.



Esempio: La figura qui sotto a sinistra rappresenta una griglia con indicati i valori delle monete presenti. Nelle altre figure sono evidenziati tre possibili cammini che il giocatore potrebbe seguire. I valori complessivi delle monete di cui il giocatore entra in possesso sono 33, 55 e 24, rispettivamente.



Si vuole costruire un algoritmo che data la griglia con le monete, determini il *valore complessivo massimo* delle monete di cui il giocatore può entrare in possesso.

Cosa si richiede: Supponete che l'input del problema sia rappresentato da una matrice M di dimensione $n \times n$, contenente numeri interi. Se nel quadrato di riga i e colonna j vi è una moneta, l'elemento $m_{i,j}$ della matrice è il valore di tale moneta, altrimenti è 0.

Indichiamo con $c_{i,j}$ il massimo valore delle monete di cui il giocatore può essere in possesso quando raggiunge il quadrato (i, j) essendo partito da un qualunque quadrato della colonna di sinistra (cioè la numero 1). Pertanto $c_{i,1} = m_{i,1}$, per $i = 1, \ldots, n$.

Qui a fianco sono mostrate la matrice M e la matrice C contenente i valori $c_{i,j}$ calcolati per l'esempio precedente.

$$M = \begin{bmatrix} 7 & 0 & 10 & 0 & 0 \\ 0 & 4 & 0 & 12 & 0 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 8 & 0 & 50 & 0 \\ 0 & 0 & 0 & 16 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 7 & 7 & 21 & 21 & 21 \\ 0 & 11 & 11 & 33 & 33 \\ 0 & 0 & 0 & 0 & 55 \\ 0 & 8 & 8 & 50 & 50 \\ 0 & 0 & 0 & 24 & 24 \end{bmatrix}$$

(a) Tenendo conto delle regole con cui il giocatore si può muovere, scrivete una o più formule per il calcolo dei valori $c_{i,j}$ corrispondenti alla colonna j di C, sulla base dei valori corrispondenti alla colonna $j-1$.
Indicate come ottenere il valore complessivo massimo delle monete di cui il giocatore può entrare in possesso, dai valori $c_{i,n}$.

(b) Descrivete *sinteticamente a parole* e poi *ad alto livello* in pseudocodice, un algoritmo basato sulla tecnica di *programmazione dinamica* che risolva il problema.

(c) Fornite una stima del tempo totale utilizzato dall'algoritmo in funzione di n, giustificando la risposta.

(d) Indicate come e in quanto tempo, una volta ottenuta la matrice C sia possibile ricavare un cammino nella matrice che permetta al giocatore di raggiunge l'obiettivo.

Note

- I punti precedenti ***devono essere risolti nell'ordine indicato.***
- ***Non cambiate i nomi*** delle matrici stabiliti nel testo dell'esercizio.
- Gli indici delle matrici sono da 1 a n. Osservate che per la riga di indice i, la riga immediatamente superiore (se $i > 1$) ha indice $i - 1$, mentre la riga immediatamente inferiore (se $i < n$) ha indice $i + 1$.
- **Attenzione al significato di righe e colonne e all'ordine con cui vanno riempite nella matrice C!**

IMPORTANTE: TUTTE le risposte devono essere adeguatamente giustificate.

Cosa si richiede: Supponete che l'input del problema sia rappresentato da una matrice M di dimensione $n \times n$, contenente numeri interi. Se nel quadrato di riga i e colonna j vi è una moneta, l'elemento $m_{i,j}$ della matrice è il valore di tale moneta, altrimenti è 0.

Indichiamo con $c_{i,j}$ il massimo valore delle monete di cui il giocatore può essere in possesso quando raggiunge il quadrato (i, j) essendo partito da un qualunque quadrato della colonna di sinistra (cioè la numero 1). Pertanto $c_{i,1} = m_{i,1}$, per $i = 1, \ldots, n$.

Qui a fianco sono mostrate la matrice M e la matrice C contenente i valori $c_{i,j}$ calcolati per l'esempio precedente.

M

$$\begin{bmatrix} 7 & 0 & 10 & 0 & 0 \\ 0 & 4 & 0 & 12 & 0 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 8 & 0 & 50 & 0 \\ 0 & 0 & 0 & 16 & 0 \end{bmatrix}$$

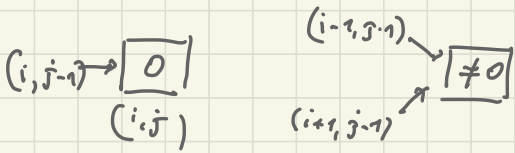
C

$$\begin{bmatrix} 7 & 7 & 21 & 21 & 21 \\ 0 & 11 & 11 & 33 & 33 \\ 0 & 0 & 0 & 0 & 55 \\ 0 & 8 & 8 & 50 & 50 \\ 0 & 0 & 0 & 24 & 24 \end{bmatrix}$$

(a) Tenendo conto delle regole con cui il giocatore si può muovere, scrivete una o più formule per il calcolo dei valori $c_{i,j}$ corrispondenti alla colonna j di C, sulla base dei valori corrispondenti alla colonna $j - 1$.
Indicate come ottenere il valore complessivo massimo delle monete di cui il giocatore può entrare in possesso, dai valori $c_{i,n}$.

- $c_{i1} = m_{i1}$ $\quad i = 1 \ldots n$ $\quad$ 1ª colonna

- colonne successive



$$c_{ij} = \begin{cases} c_{i,j-1} & \text{se } m_{ij} = 0 \\ m_{ij} + \max\left(c_{i-1,j-1}, c_{i+1,j-1}\right) & \end{cases}$$

$j > 1$
$i \in \{1, \ldots, n\}$

$c_{i-1,j-1}$: escluso $i = 1$ $\quad$ $c_{i+1,j-1}$: escluso $i = n$

ricordare: $\max \{c_{in},\ i = 1 \ldots n\}$

(b) Descrivete *sinteticamente a parole* e poi *ad alto livello* in pseudocodice, un algoritmo basato sulla tecnica di *programmazione dinamica* che risolva il problema.

(c) Fornite una stima del tempo totale utilizzato dall'algoritmo in funzione di n, giustificando la risposta.

ALGORITMO cammino (Matrice $M[1..n, 1..n]$) → intero

Sia $C[1..n, 1..n]$ una matrice

FOR $i \leftarrow 1$ TO n DO

  $C[i, 1] \leftarrow M[i, 1]$

$$C_{ij} = \begin{cases} C_{i,j-1} & \text{se } m_{ij} = 0 \\ m_{ij} + \max(C_{i-1,j-1}, C_{i+1,j-1}) & \end{cases}$$

$j > 1$, $i \in [1..n]$

$C_{i-1,j-1}$: esiste se $i > 1$; $C_{i+1,j-1}$: esiste se $i < n$

FOR $j \leftarrow 2$ TO n DO

  FOR $i \leftarrow 1$ TO n DO

    IF $M[i,j] = 0$ THEN $C[i,j] \leftarrow C[i,j-1]$

    ELSE

      MAX $\leftarrow 0$

      IF $i > 1$ AND $C[i-1, j-1] >$ MAX THEN MAX $\leftarrow C[i-1, j-1]$

      IF $i < n$ AND $C[i+1, j-1] >$ MAX THEN MAX $\leftarrow C[i+1, j-1]$

      $C[i,j] \leftarrow M[i,j]$ + MAX

MAX $\leftarrow C[1, n]$

FOR $i \leftarrow 2$ TO n DO

  IF $C[i, n] >$ MAX

  THEN MAX $\leftarrow C[i, n]$

RETURN max

(d) Indicate come e in quanto tempo, una volta ottenuta la matrice C sia possibile ricavare un cammino nella matrice che permetta al giocatore di raggiunge l'obiettivo.