Algoritmi e Strutture Dati

Lezione 36

17 dicembre 2025

Cognome . . . . . .

Nome . . . . . .

Matricola . . . . . .

Algoritmi e Strutture Dati

Prova scritta del 14 giugno 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 `24 17 43 14 12 26 39 34 16`.

(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 seguente sequenza di numeri, memorizzata in un array: `24 17 43 14 12 26 39 34 16`.

(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 in cui la chiave `39` raggiunge la sua posizione definitiva.

(c) Supponete di ordinare la sequenza *in modo crescente* mediante l'algoritmo `heapSort`.
Indicate il contenuto dell'array immediatamente dopo lo scambio che colloca `39` nella sua posizione definitiva.

(d) Supponete di ordinare la sequenza *in modo decrescente* mediante l'algoritmo `heapSort`.
Indicate il contenuto dell'array dopo averlo trasformato in uno heap.

3. Per definire le tasse sugli immobili, gli amministratori di una città hanno deciso di suddividere il territorio in due aree, chiamate *centrale* e *periferica*. Essendo l'area centrale di maggior pregio, in essa la tassazione sarà più elevata.

Le aree sono individuate in base alla distanza dal centro. Come *distanza di un luogo dal centro*, si intende la lunghezza del più breve percorso che permette di raggiungere il luogo partendo dal centro. Indicando con r la distanza del luogo della città più lontano dal centro, le aree sono definite come segue:

- l'*area centrale* è costituita dai luoghi che si trovano a distanza $d \leq \frac{1}{2}r$ dal centro,
- l'*area periferica* è costituita dai luoghi che si trovano a distanza $d > \frac{1}{2}r$ dal centro.

Cosa di richiede

Supponete che la mappa della città sia descritta da un grafo orientato i cui vertici rappresentano gli incroci tra le strade (anche la fine di una strada a fondo chiuso è considerata un incrocio). Gli archi rappresentano tratti di strada, delimitati da due incroci. Ad ogni arco è associato come peso la lunghezza del tratto di strada rappresentato. Il centro della città si trova in un particolare vertice del grafo, indicato con s. Pertanto il valore di r è la massima distanza da s di un vertice del grafo.

Progettate un algoritmo che permetta di individuare tutti gli incroci che appartengono all'area centrale. Pertanto, l'algoritmo:

- riceve in *input* il grafo orientato pesato $G = (V, E, \omega)$, che descrive la mappa stradale della città, e il vertice $s \in V$ che indica il centro,
- fornisce in *output* un elenco contenente tutti i vertici che rappresentano gli incroci dell'area centrale.

Risolvete i seguenti punti ***nell'ordine indicato.***

(a) Descrivete sinteticamente a parole la strategia dell'algoritmo e scrivete il suo pseudocodice *esclusivamente ad alto livello.* È opportuno ispirarsi a uno degli algoritmi presentati nel corso (indicate quale).

(b) Discutete una possibile rappresentazione del grafo utilizzato e una corrispondente implementazione dell'algoritmo, ricorrendo anche a eventuali strutture di supporto (se lo ritenete utile, potete scrivere lo pseudocodice dell'algoritmo includendo le operazioni su tali strutture), fornendo una stima dei tempi di calcolo.
Giustificate dettagliatamente la stima dei tempi di calcolo.

(c) Supponete che si debba determinare l'elenco dei tratti di strada che si trovano tra le due aree, cioè che collegano un incrocio dell'area centrale con un incrocio dell'area periferica. Indicate come potete modificare l'algoritmo che avete progettato nei punti precedenti a questo scopo e fornite una stima del tempi di calcolo delle nuove parti che avete aggiunto.

Le risposte a questo esercizio devono essere adeguatamente giustificate.

(a) Descrivete sinteticamente a parole la strategia dell'algoritmo e scrivete il suo pseudocodice *esclusivamente ad alto livello*. È opportuno ispirarsi a uno degli algoritmi presentati nel corso (indicate quale).

- algoritmo di Dijkstra: calcolo distanze
- calcolo di r
- creazione dell'elenco da potature in output (ispezionando tutti i vertici)

```
ALGORITMO areaCentrale(Grafo G=(V,E,w), vertice s) → Elenco
    Sia d[v] un array con indici in V
    d[s] ← 0
    FOR EACH v ∈ V−{s} DO  d[v] ← ∞
    C ← V                                  ] creazione e riempimento coda con priorità
    WHILE C ≠ ∅ DO                         n inserimenti
        x ← elemento in C con d[x] minimo  → x ← C.deleteMin()
        C ← C−{x}                          r ← d[x]
        FOR EACH (x,y) ∈ E DO              costo totale m
            IF d[x] + w(x,y) < d[y]
            THEN [d[y] ← d[x] + w(x,y)
                  C.changeKey(y, d[y])     costo singolo O(log n) al più m volte
    r ← 0
    FOR EACH x ∈ V DO
        IF d[x] > r THEN r ← d[x]

    Elenco ← ∅
    FOR EACH x ∈ V DO                      ← n iterazioni
        IF d[x] ≤ (1/2) r THEN
            Elenco ← Elenco ∪ {x}          costo con inserimento alla fine O(1)
    RETURN Elenco
```

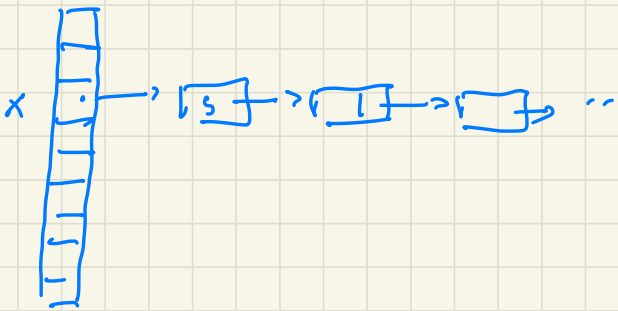
costo singolo $O(\log n)$, n volte $O(n \log n)$

costo totale $O(m)$

$O(n \log n) + O(m) + O(m \log n) + O(n)$ $\quad O(n)$

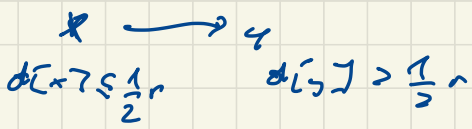
(b) Discutete una possibile rappresentazione del grafo utilizzato e una corrispondente implementazione dell'algoritmo, ricorrendo anche a eventuali strutture di supporto (se lo ritenete utile, potete scrivere lo pseudocodice dell'algoritmo includendo le operazioni su tali strutture), fornendo una stima dei tempi di calcolo. *Giustificate dettagliatamente la stima dei tempi di calcolo.*

rappresentazione: lista di adiacenza / incidenza



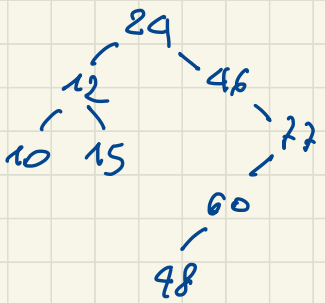
struttura di supporto: Coda con priorità che associa ad ogni vertice v il valore d[v]

(c) Supponete che si debba determinare l'elenco dei tratti di strada che si trovano tra le due aree, cioè che collegano un incrocio dell'area centrale con un incrocio dell'area periferica. Indicate come potete modificare l'algoritmo che avete progettato nei punti precedenti a questo scopo e fornite una stima del tempi di calcolo delle nuove parti che avete aggiunto.

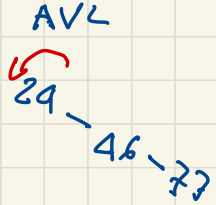


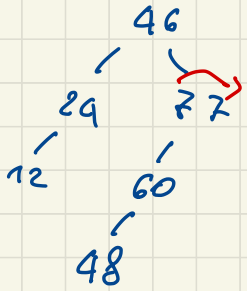
Data la sequenza di numeri: 24 46 77 12 60 48 10 15 disegnare l'albero di binario si ricerca, l'albero AVL e l'albero 2-3 ottenuti inserendo in un albero inizialmente vuoto i numeri della sequenza, uno dopo l'altro, nell'ordine indicato

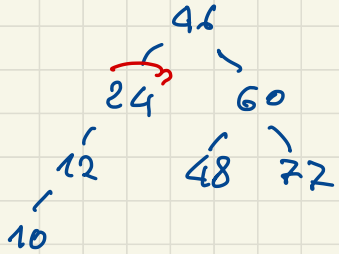
albero bin ricerca

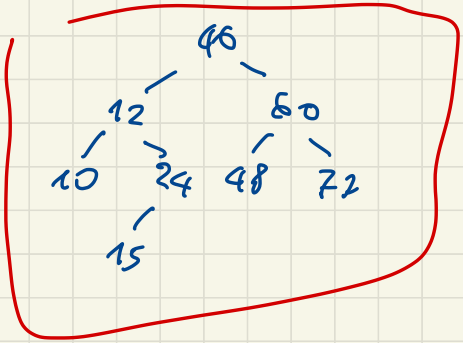


AVL

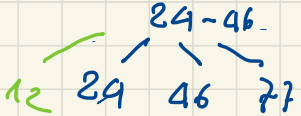


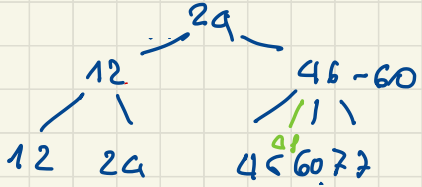


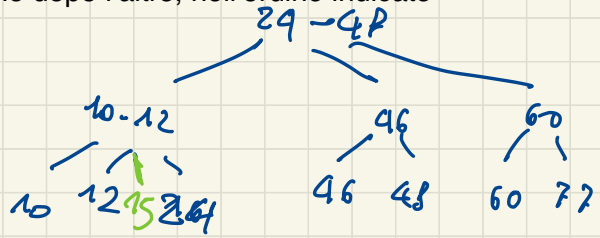


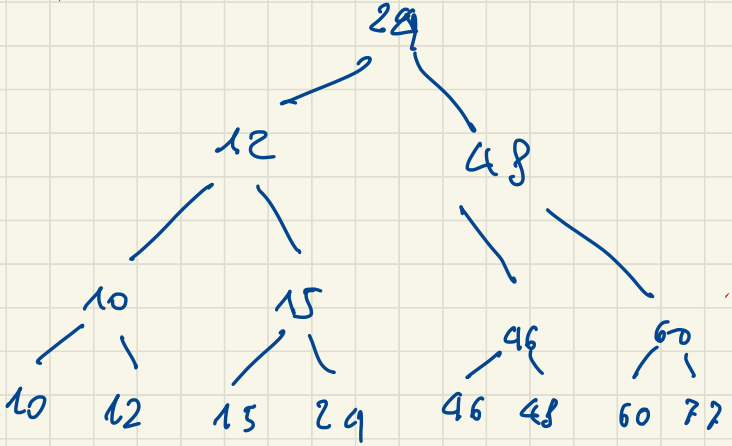


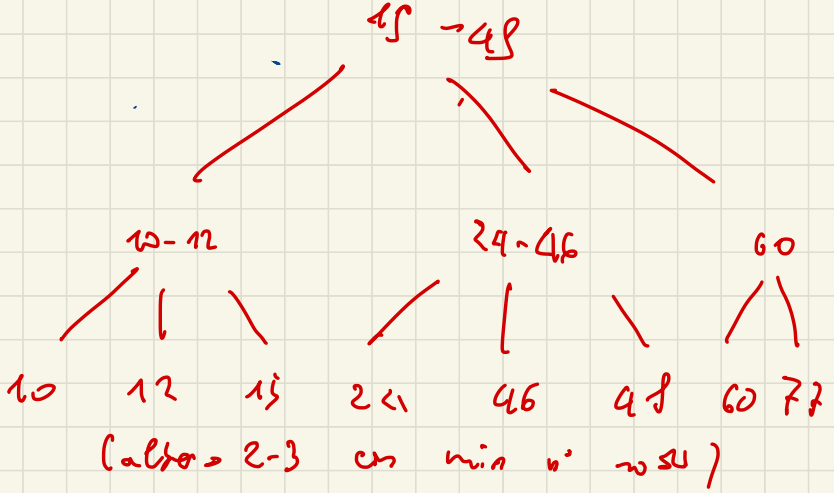
Data la sequenza di numeri: 24 46 77 12 60 48 10 15 disegnare l'albero di binario si ricerca, l'albero AVL e l'albero 2-3 ottenuti inserendo in un albero inizialmente vuoto i numeri della sequenza, uno dopo l'altro, nell'ordine indicato



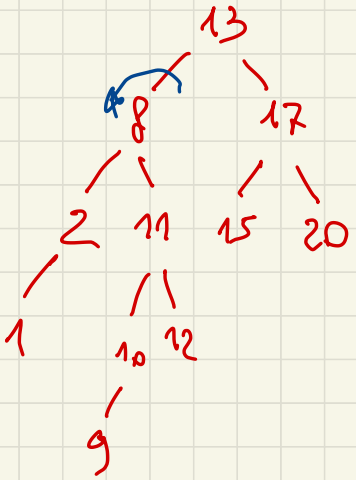


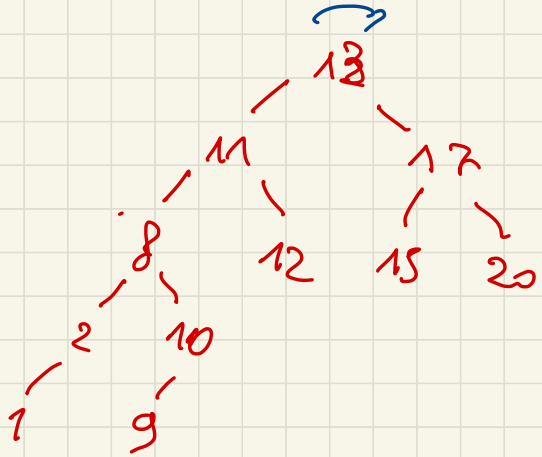


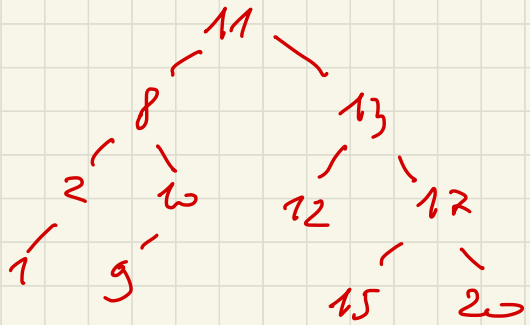




Data la sequenza di numeri 13 17 8 2 20 15 11 12 10 1 9 disegnate l'albero di binario si ricerca, l'albero AVL e l'albero 2-3 ottenuti inserendo in un albero inizialmente vuoto i numeri della sequenza, uno dopo l'altro, nell'ordine indicato







2. Considerate la seguente sequenza di numeri, memorizzata in un array: `22 15 41 12 10 24 37 32 14`.

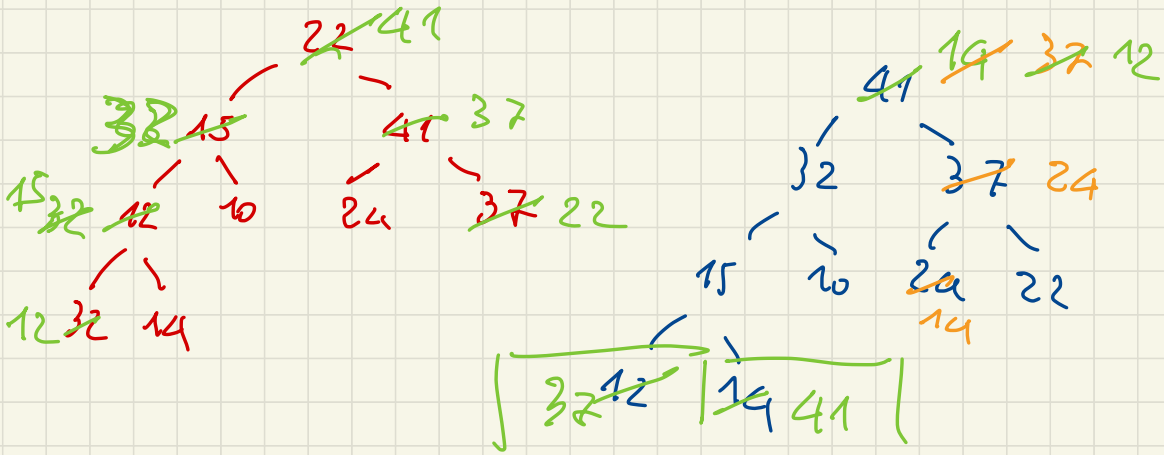
(a) Supponete di ordinare la sequenza *in modo crescente* mediante l'algoritmo `quickSort`, scegliendo come perno il primo elemento ed utilizzando, per effettuare la partizione, la strategia presentata a lezione.
Indicate il contenuto dell'array dopo avere effettuato la partizione, prima delle chiamate 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 in cui la chiave `37` raggiunge la sua posizione definitiva.

(c) Supponete di ordinare la sequenza *in modo crescente* mediante l'algoritmo `heapSort`.
Indicate il contenuto dell'array immediatamente dopo lo scambio che colloca `37` nella sua posizione definitiva.

12 32 24 15 10 14 22 37 41

(d) Supponete di ordinare la sequenza *in modo decrescente* mediante l'algoritmo `heapSort`.
Indicate il contenuto dell'array dopo averlo trasformato in uno heap.



2. Considerate la seguente sequenza di numeri, memorizzata in un array: `22 15 41 12 10 24 37 32 14`.

(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 in cui la chiave `37` raggiunge la sua posizione definitiva.

(c) Supponete di ordinare la sequenza *in modo crescente* mediante l'algoritmo `heapSort`.
Indicate il contenuto dell'array immediatamente dopo lo scambio che colloca `37` nella sua posizione definitiva.

(d) Supponete di ordinare la sequenza *in modo decrescente* mediante l'algoritmo `heapSort`.
Indicate il contenuto dell'array dopo averlo trasformato in uno heap.