Algoritmi e Strutture Dati Laboratorio A.A 2021/2022

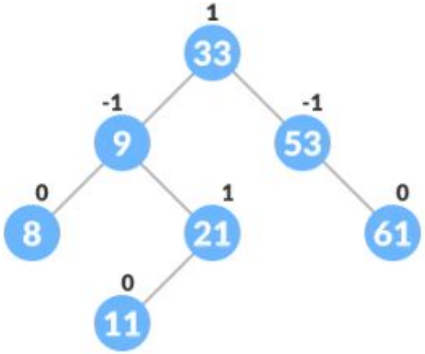
Carmine Marchesani - Matricola : 113916
Patryk Sebastian Bialowas - Matricola : 113959

carmine.marchesani@studenti.unicam.it
patryk.bialowas@studenti.unicam.it

Complessità

- Complessità temporale : nell’operazione di inserimento di un elemento in un AVL Tree, la complessità temporale in tutti e **3 i casi** (migliore, medio e peggiore) è di ***O(log n)*.**
- Complessità dello spazio: in tutti e 3 i casi è di ***O(n)***

Alberi AVL



- Complessità temporale : nell'operazione di inserimento di un elemento in un AVL Tree, la complessità temporale in tutti e **3 i casi** (migliore, medio e peggiore) è di ***O(log n)*.**

- Complessità dello spazio: in tutti e 3 i casi è di ***O(n)***

- insert( )
- rotazioni
  - destra / sinistra
  - destra / destra
  - sinistra / destra
  - sinistra / sinistra

AVL Tree Sort

- Complessità **temporale** :

  -

  Caso migliore: **O(n log n)**

  -**Caso medio**: **O(n log n)**

  -**Caso peggiore**: **$O(n^2)$** se sbilanciato

  O(n log n) se bilanciato

- Complessità dello spazio:

  Θ(n)

- sort()

Heap Sort

L’algoritmo di Heapsort si occupa di ordinare gli elementi dell’albero servendosi di **due principali metodi**.

In questo caso è stato fatto l’ordinamento con la variante che consiste nel nodo padre che ha 3 figli, quindi **heap ternario,** invece di 2 come nel classico heap **binario**.

- buildMaxHeap()
- heapify()

Counting Sort

Il **Counting sort** è un algoritmo di ordinamento per valori numerici interi con complessità lineare. L'algoritmo si basa sulla conoscenza a priori dell'intervallo in cui sono compresi i valori da ordinare.

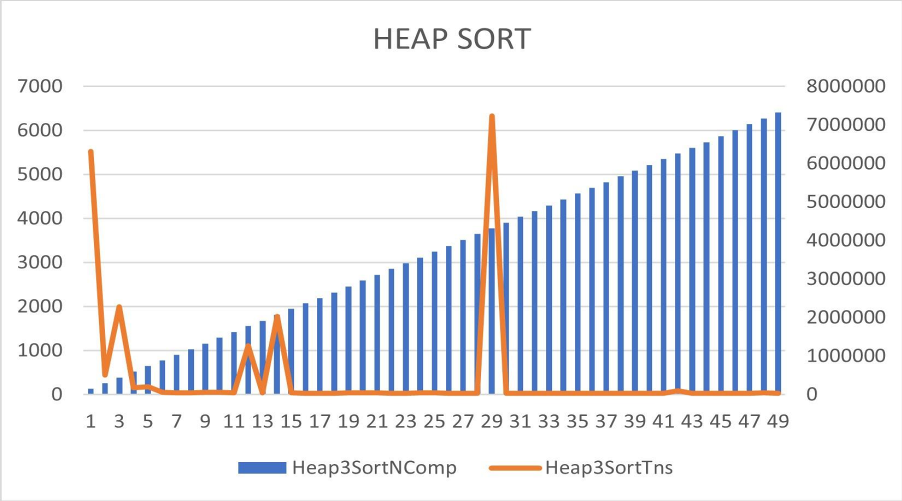
L'algoritmo conta il numero di occorrenze di ciascun valore presente nell'array da ordinare

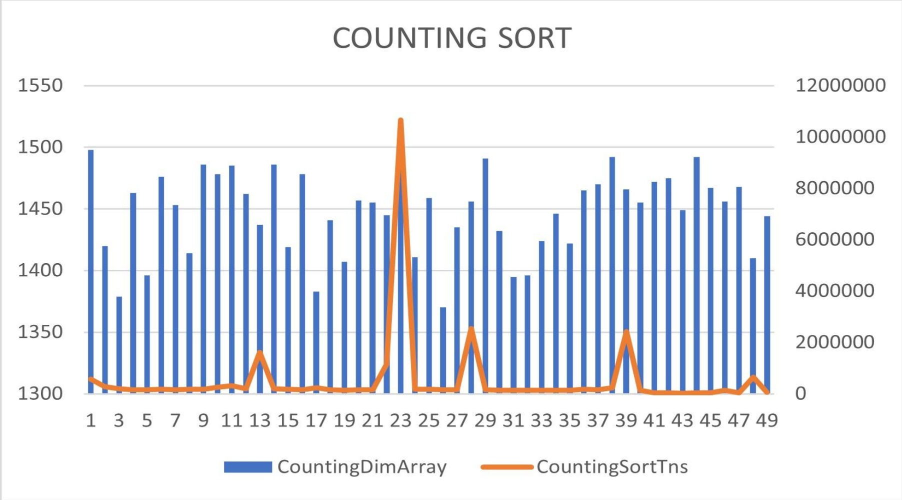
- sort()

Dati dei grafici generati

Complessità HeapSort:
Caso **migliore**, caso **medio** e caso **peggiore**:
O(n log n)

Complessità **Counting Sort:**
Caso **migliore**, caso **medio** e caso **peggiore**:
Andamento **non lineare**:
T(n) = O(k) + O(n) = **O(k + n)**
Andamento **lineare**:
T(n) = O(2n) = **O(n)**





Matrice di adiacenza

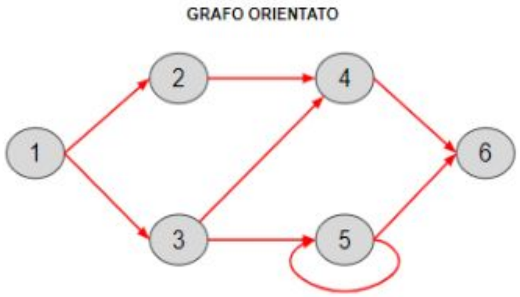
Grafo : una struttura matematica che consiste in un insieme di archi e vertici.

La matrice delle **adiacenze** o *matrice di* ***connessione*** costituisce una particolare struttura dati comunemente utilizzata nella rappresentazione dei grafi **finiti**.

Nella matrice di adiacenza un grafo viene rappresentato mediante una matrice quadrata **M** di dimensione **n × n** (con n = |V|, il numero di nodi del grafo) in cui Mi,j = 1 **se (vi , vj) ∈ E**, Mi,j = 0 altrimenti.

Adjacency Matrix Directed Graph

Esempio di **matrice di adiacenza** contenente gli archi che collegano i vari nodi.



nodi destinazione

nodi di partenza	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	1	0	0
3	0	0	0	1	1	0
4	0	0	0	0	0	1
5	0	0	0	0	1	1
6	0	0	0	0	0	0

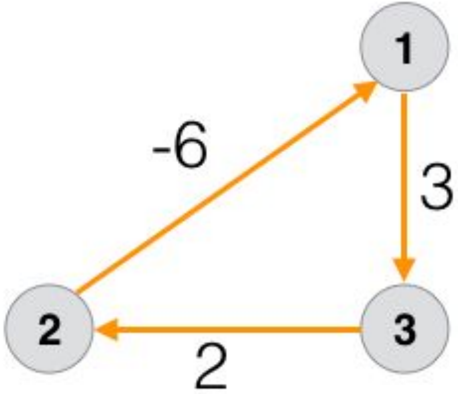
- addNode()
- removeNode()
- addEdge()
- removeEdge()

Algoritmo di Bellman Ford

L'algoritmo di Bellman Ford risolve il problema dei cammini minimi tra nodi di un grafo da **sorgente unica**.

Questo algoritmo può contenere pesi negativi, ma non **cicli** di **peso negativo.**

Esempio di ciclo di peso negativo



Bellman Ford

- Complessità **temporale** :
  Caso migliore: **O(E)**
  -**Caso medio**: **O(E V)**
  -**Caso peggiore**: **O(V^3)**
- Complessità dello **spazio**:

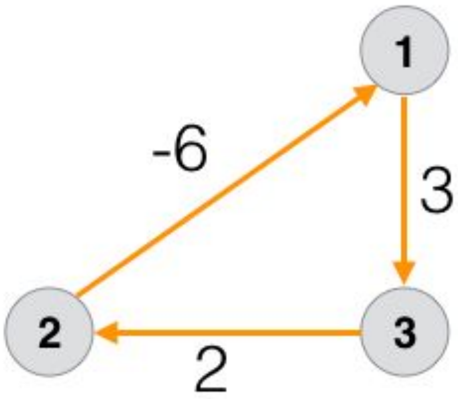
-**O(V^2)** per una matrice di adiacenza
-**O(V)** per una lista con lunghezza V

- computeShortestPathFrom()
- getShortestPathTo()

Algoritmo di Floyd Warshall

L'**algoritmo di Floyd-Warshall** calcola il cammino minimo per tutte le coppie di un grafo pesato e orientato.

Questo algoritmo può contenere pesi negativi, ma non **cicli** di **peso negativo.**
Esempio di ciclo di peso negativo



Floyd Warshall

- Complessità **temporale** :

  O(V^3).

  Dove V rappresenta il numero di vertici del grafo.

- Complessità nello **spazio** :

  O(n).

  L' algoritmo è complementare a quello di Bellman, con la differenza che è più stabile.

- computeShortesPaths()
- getShortestPath()
- getShortestPathCost()

Grazie mille
per l’attenzione!

“divide et impera”

(Programmazione dinamica)