Algoritmi e Strutture Dati
(SSRI Crema)

Appunti, Esercizi e Risposte ai Temi di Esami dal 2014 al 2017.
Raccolta focalizzata per studenti Online.

Andrea Draghetti

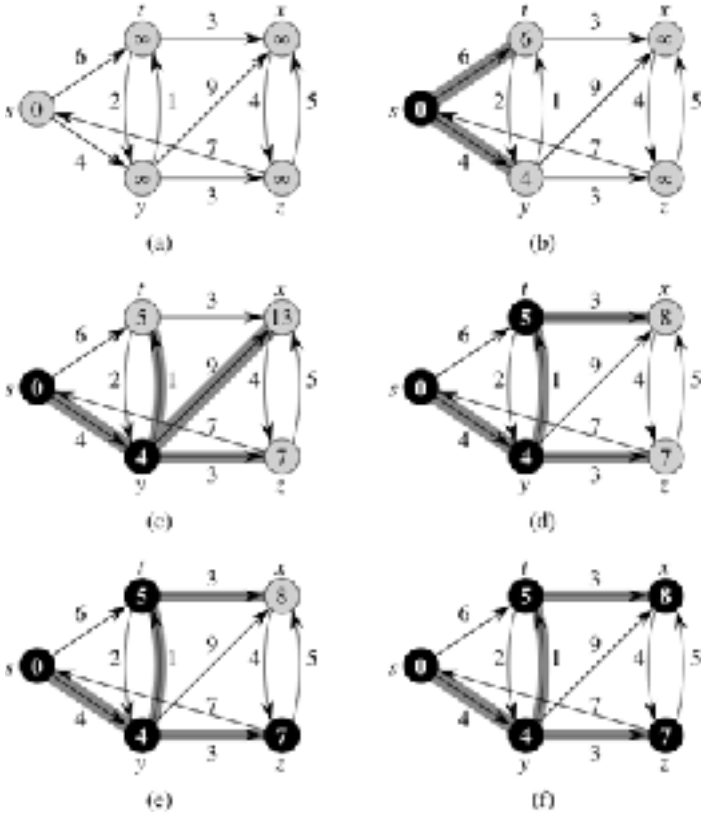
Consigli:
Dall'anno 2014 per gli studenti Online è cambiata la prova di esame, dove prima erano presenti solo esercizi attualmente vengono solitamente proposti 10 domande di teoria e 3 esercizi. Le video lezioni attuali, al 2017, sono ancora quelle del precedente docente Roberto Aringhieri che non segue e ne copre completamente gli argomenti trattati dalla Sabrina De Capitani di Vimercati. Si consiglia pertanto di guardare le slide e le registrazioni delle lezioni in presenza.

Altro materiale disponibile su www.swappa.it, non si garantisce la correttezza dei seguenti appunti. Per maggiore sicurezza fare riferimento ai libri di testo proposti o alle slide delle lezioni.

Dijkstra

- **A cosa serve l'algoritmo di Dijkstra? Quale è la sua complessità?**

L'algoritmo Dijkstra viene sfruttato per cercare i cammini minimi in un grafo con pesi non negativi sugli archi. La sua complessità è O(n^2).



Kruskal

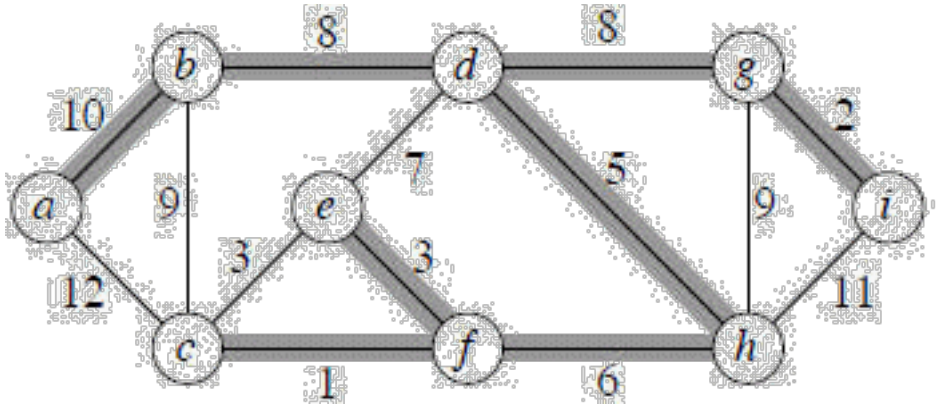
- **A cosa serve l'algoritmo di Kruskal? Quale è la sua complessità?**

L'algoritmo di Kruskal viene sfruttato per calcolare gli alberi di copertura minimi di un grafo non orientato e con gli archi con costi non negativi.

Si consideri un grafo non orientato e connesso dove V rappresenta il numero di vertici (o nodi) ed E il numero di spigoli (o archi). Ad ogni spigolo è associato un peso (o distanza): lo scopo dell'algoritmo è quello di trovare un albero ricoprente di peso minimo, cioè quello in cui la somma dei pesi sia minima. L'algoritmo può essere applicato solo se si dispone di due o più vertici.

L'algoritmo di Kruskal si basa sulla seguente semplice idea: ordiniamo gli archi in ordine crescente di costo e successivamente li analizziamo singolarmente, inseriamo l'arco nella soluzione se non forma cicli con gli archi precedentemente selezionati. Notiamo che ad ogni passo, se abbiamo più archi con lo stesso costo, è indifferente quale viene scelto.

La complessità è O(m log n).



Moore

- **A cosa serve l'algoritmo di Moore? Quale è la sua complessità temporale?**

Viene utilizzato per risolvere problemi di scheduling, in cui si hanno dei programmi da eseguire su un processore e si vuole trovare l'ordine di esecuzione ottimo in base ad un prefisso criterio. Ovvero dato il seguente problema: dati n programmi p tali che ciascun richiede unità di tempo per la sua esecuzione e deve essere completato entro una certa scadenza, trovare un ordine S in cui eseguire tutti i programmi in modo da minimizzare il numero di programmi per i quali la scadenza non è rispettata. L'algoritmo di Moore risolve questo tipo di problema.Una realizzazione poco accorta dell'algoritmo porta ad una complessità di O(n^2), ma può essere migliorata attraverso l'uso di una coda di priorità Q implementata con uno heap modificato (per trattare programmi), ottenendo una complessità O(n log n).

Nel caso pessimo avremo O(n) eliminazioni => algoritmo è O(n^2)

Pape-D'Esopo

- **A cosa serve l'algoritmo di Pape-D'Esopo? Cosa si intende per grafo planare? Quale è la sua complessità?**

L'algoritmo di Pape-D'Esopo, per il calcolo dei cammini minimi, usa come insieme S una dequeue (double ended queue). Una dequeue è una coda che ammette anche l'inserzione in testa, e non solo in coda. Il generico nodo u verrà inserito la prima volta in coda ed in testa le volte successive. Più precisamente, se d[u] == MAXINT, allora u è inserito in coda, altrimenti in testa. L'idea dell'inserimento in testa è quella di sfruttare immediatamente il miglioramento dell'etichetta affinché esso si propaghi ai nodi vicini. L'algoritmo, nel caso pessimo, ha complessità superpolinomiale!! Tuttavia, si è sperimentalmente verificato essere uno dei più efficienti su grafi che modellano reti di comunicazione stradali, ovvero su grafi sparsi e planari. Un grafo è planare quando, disegnato su un piano, le linee corrispondenti a due archi distinti non si sovrappongono mai.

Albero Binario di Ricerca Bilanciato (AVL)

- **Come è definito il fattore di bilanciamento di un nodo in un albero AVL?**

Il fattore di bilanciamento di un nodo v è la massima differenza di altezza fra i sottoalberi di v, l'albero perfetto è un albero dove il fattore di bilanciamento è 0 per ogni nodo v.

Albero Binario di Ricerca

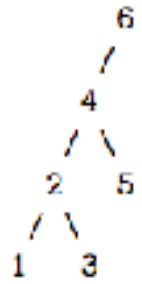
- **Cos'è un Albero Binario di Ricerca?**

Gli alberi binari di ricerca (ABR) sono strutture dati che possono eseguire molte operazioni su insiemi dinamici come: SEARCH, MINIMUM, MAXIMUM, PREDECESSOR, SUCCESSOR, INSERT, DELETE. Possono essere utilizzati sia come dizionario che come code di priorità. Il tempo delle operazioni è proporzionale all'altezza dell'albero. Ogni nodo può aver al massimo due figli rappresentati da record. Un ABR è rappresentato da una struttura dati concatenata in cui ogni nodo è un oggetto. Ogni nodo ha i seguenti campi: Puntatore al Padre, Figlio Sx, Figlio Dx, Chiave e Dati. La mancanza di un figlio è rappresentata con puntare a NIL. La radice è l'unico nodo ad avere puntatore al padre NIL.

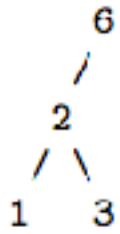
Le proprietà da rispettare sono: tutti i valori dei nodi del sottoalbero sinistro sono minori della radice mentre quelli del sottoalbero destro sono maggiori della radice. La visita si può eseguire in profondità con uno dei 3 algoritmi di ritorsione: preordine, inordine, postordine

- **Dato un albero binario di ricerca, l'operazione di cancellazione su tale albero è commutativa? Si richiede di giustificare la risposta mostrando anche un esempio.**

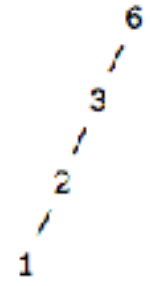
Per commutativa si intende che la cancellazione del valore x e poi del valore y produca lo stesso risultato che rimuovere prima il valore y e poi x. Negli ABR tale operazione NON è commutativa. Esempio:



Rimuovendo prima il valore 5, poi il valore 4 si ottiene questo risultato:



Se invece rimuovo prima il valore 4, poi 5 ottengo questo risultato diverso:



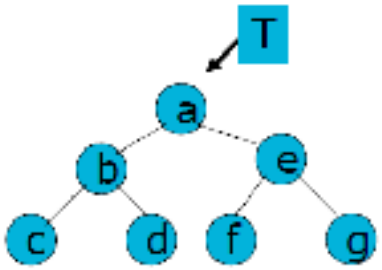
- **Dato un albero binario T, si richiede di descrivere le visite di tipo preordine, postordine ed inordine su T.**

Visita in preordine: Si visita prima la radice e poi si effettuano le chiamate ricorsive sul figlio sinistro e destro; in altre parole l'esportazione dell'albero parte dalla radice per poi scendere alle foglie, che sono gli ultimi nodi ad essere visitati.

Visita in ordine (simmetrica): Si effettua prima la chiamata ricorsiva sul figlio sinistro, poi si visita la radice ed infine si effettua la chiamata ricorsiva sul figlio destro. In sostanza per ogni nodo si visita prima il sottoalbero sinistro poi si visita il nodo corrente ed infine si passa al sottoalbero destro.

Visita in postordine: Si effettuano prima le chiamate ricorsive sul figlio sinistro e destro e poi si visita la radice; in altre parole la visita dell'albero parte dalle foglie per poi risalire alla radice, che è l'ultimo nodo ad essere esplorato.

Tutte e tre le varianti sono algoritmi usati per l'esportazione in profondità dei nodi di un albero.



Visita in preordine o previsita: a, b, c, d, e, f, g
Visita inordine o invisita: c, b, d, a, f, e, g
Visita in postordine o postvisita: c, d, b, f, g, e, a

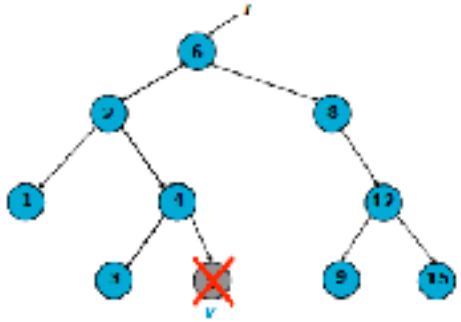
- **Descrivere il funzionamento della operazione di cancellazione su un albero binario di ricerca. Si richiede inoltre di mostrare un esempio per tutti i casi che si possono verificare.**

La cancellazione di un elemento in un Albero Binario di Ricerca (ABR) non è una operazione immediata. Per mantenere le proprietà dell'ABR anche dopo la cancellazione infatti bisogna distinguere il caso in cui il nodo da cancellare sia una foglia senza figli, oppure che abbia un figlio o due figli. Ci possono quindi essere tre casi:

1) Se il nodo è una foglia senza figli, basta cancellarlo dall'albero.
2) Se il nodo invece ha un figlio solo, si elimina sostituendolo nella struttura dell'albero con il suo unico figlio.
3) Se il nodo invece ha due figli si ricerca il suo successore, e si scambia i valori del nodo da cancellare e del successore trovato, cancellando poi solo quest'ultimo (che ha zero o un figlio solo).

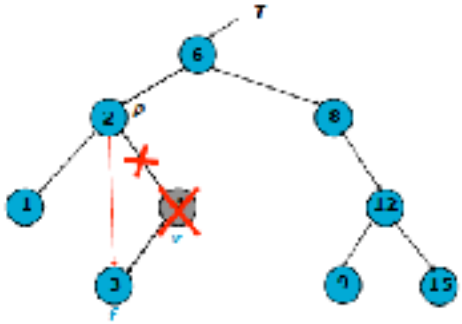
Cancellazione: Caso 1

Il nodo v da cancellare non ha figli



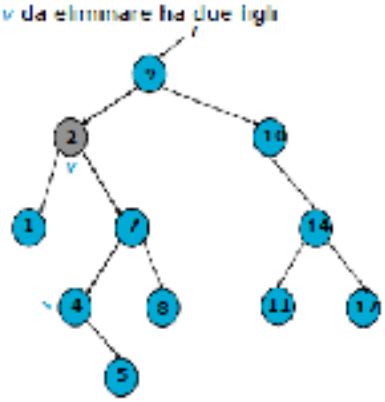
Cancellazione: Caso 2

Il nodo v da cancellare ha un solo figlio f



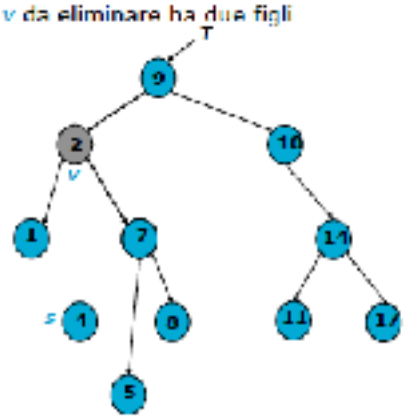
Cancellazione: Caso 3 (1)

Il nodo v da eliminare ha due figli



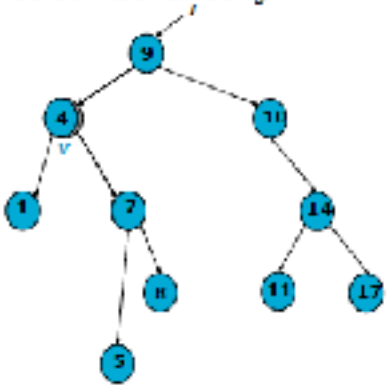
Cancellazione: Caso 3 (2)

Il nodo v da eliminare ha due figli



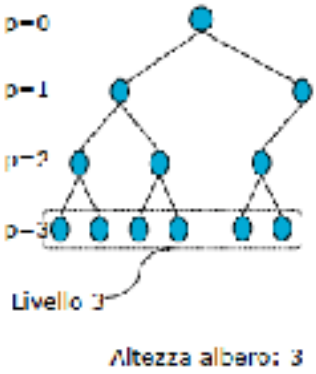
Cancellazione: Caso 3 (3)

Il nodo v da eliminare ha due figli



- **In un albero binario completo di altezza h quanti sono i nodi foglia? Giustificare la risposta.**

Numero Massimo Nodi: $2^{(h+1)}-1$
Numero Massimo Nodi Foglia: 2^h



- **Quale è la complessità computazionale per la ricerca del valore minimo di un albero binario di ricerca?**

L'algoritmo per la ricerca di un elemento ha complessità algoritmica O(h) dove h è la profondità dell'albero.

L'algoritmo per la visita ordinata ha complessità algoritmica O(n).

Hash

- **Nell'ambito delle tabelle di hash, descrivere i metodi di scansione linerare, quadratica e hashing doppio.**

Se si implementa un dizionario con tabelle di Hash si possono verificare delle collisioni. Una collisione nasce quando due chiavi k1 e k2 vengono mappata dalla funzione hash H nella stessa cella. Ovvero H(k1) == H(k2).

I metodi per la gestione delle collisioni sono i seguenti:
- Scansione lineare: Quando viene incontrata una collisione non si fa altro che utilizzare l'indice successivo a quello che collide, fino a che non si trovi una cella libera.
- Scansione quadratica: Quando viene incontrata una collisione non si fa altro che utilizzare l'indice che collide elevato al quadrato con normalizzazione rispetto alla grandezza della tabella dell'indice ottenuto, sino a che non si trovi una casella libera.
- Hashing doppio: Se facendo l'hash di una chiave si incontra una collisione, allora si somma all'indice ottenute il risultato di una nuova funzione hash (generalmente diversa dalla prima e che ha come parametro l'indice ottenuto precedentemente), e si tenta l'inserimento nel nuovo indice così ottenuto, riapplicando la seconda funzione sino a che non si trovi una casella libera.

- **Nell'ambito delle tabelle di hash, cosa si intende per indirizzamento aperto?**

Con la tecnica di indirizzamento aperto tutti gli elementi vengono memorizzati nella tavola, la funzione di hash non individua una singola cella ma una sequenza esaustiva di esse. L'inserimento di un nuovo elemento avviene nella prima cella libertà che si incontra nella sequenza.

BackTrack

- **Cosa si intende per tecnica di Backtrack? Mostrare un esempio di algoritmo basato su tale tecnica.**

La tecnica di Backtrack viene sfruttata per la progettazione di algoritmi e si basa sul concetto di costruzione ed eventuale distruzione di parte della soluzione, più semplicemente possiamo dire che l'algoritmo prova a fare qualcosa se il risultato non è quello previsto annulla tutto e riprova.

Un algoritmo è di tipo BackTrack quando in esso sono previsti strumenti per la costruzione della soluzione che per la distruzione di parte di essa.

Questo algoritmo viene sfruttato nella tecnica di String Matching.

QuickSort

- **Cosa è il perno utilizzato nell'algoritmo Quicksort? Quale è la complessità computazionale nel caso peggiore e medio di tale algoritmo?**

Il perno, o pivot, è l'elemento chiave dell'algoritmo di Quicksort. Gli elementi minori dell'array si porranno a sinistra del pivot mentre gli elementi maggiori a destra, l'operazione di ordinamento si reitera sugli insieme risultati fino al completamento dell'ordinamento.

Nel caso pessimo il pivot è sempre elemento più piccolo, e quindi la porzione A con elementi minori di x è vuota, mentre quella con elementi maggiori o uguali ne contiene n-1. Supponiamo che questo sbilanciamento si verifichi in ogni chiamata ricorsiva. Il partizionamento costa Θ(n) in termini di tempo, applichiamo il teorema delle ricorrenze lineari di ordine costante il tempo di esecuzione è: O(n^2)

Il caso migliore si verifica quando l'algoritmo di partizionamento determina due sottoproblemi perfettamente bilanciati (ad esempio: 5,6,7,8,9,1,2,3,4), entrambi di dimensione n/2; in questo caso il tempo di esecuzione è O (n log n)

Coda

- **Definire il tipo di dato coda ed indicare sia la specifica sintattica sia la specifica semantic**a

La coda è una struttura dati di tipo FIFO (First In First Out), in cui il primo elemento inserito è il primo elemento che verrà estratto. La coda ha un inizio (head) e una fine (tail). Quando un elemento viene inserito nella coda, prende posto alla fine della coda, mentre l'elemento rimosso è sempre quello che si trova all'inizio della coda.

La specifica sintattica del tipo di dato Coda comprendono codavuota() in grado di indicare se la coda è vuota, incoda(ITEM) in grado di inserire ITEM in fondo alla cosa, fuoricoda() in grado di estrarre l'elemento in testa alla cosa e lo restituisce al chiamante e infine leggicoda() in grado di leggere l'elemento in testa alla coda.

Le specifica semantica è invece composta da codavuota(Q), leggicoda(Q), fuorica(Q), incoda(a,Q).

- **Descrivere la struttura dati Coda**

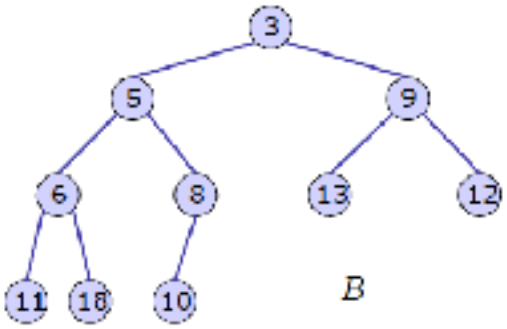
La coda è una struttura dati di tipo FIFO (First In First Out), in cui il primo elemento inserito è il primo elemento che verrà estratto. La coda ha un inizio (head) e una fine (tail). Quando un

elemento viene inserito nella coda, prende posto alla fine della coda, mentre l'elemento rimosso è sempre quello che si trova all'inizio della coda.

- **Si richiede di descrivere la realizzazione di una coda di priorità con uno heap e di mostrare un esempio sul quale si richiede anche di eseguire l'operazione di cancellamin.**

Una coda di priorità è una struttura dati che serve a mantenere un insieme S di elementi, ciascuno con un valore associato detto chiave. È simile ad una coda, ma si differisce da questa in quanto ogni elemento inserito all'interno della coda possiede una sua "priorità". In una coda di priorità, ogni elemento avente priorità più alta, viene inserito prima rispetto ad un elemento avente priorità più bassa. In particolare, l'elemento con priorità più alta si trova in testa alla coda, quello con priorità più bassa si troverà, appunto, in coda.

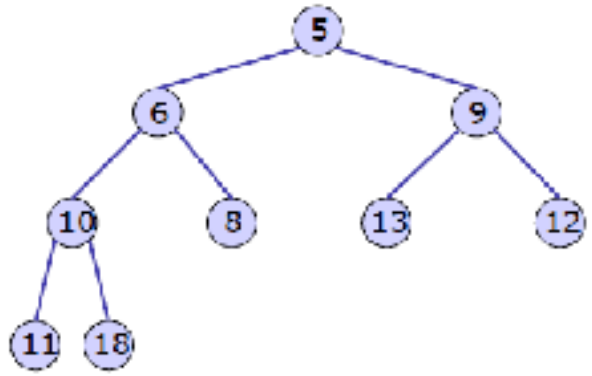
Prendiamo in esempio il seguente albero:



la relativa coda di priorità è:

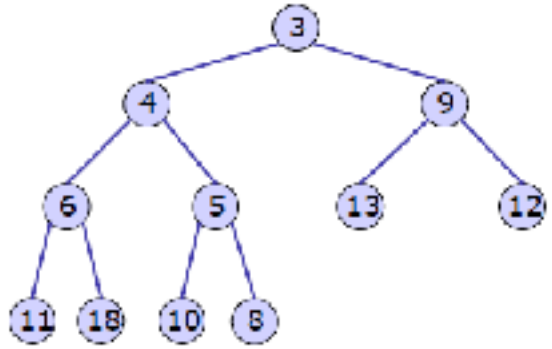
Priorità	Valore
1	3
2	5
3	9
4	6
5	8
6	13
7	12
8	11
9	18
10	10

Per eseguire "cancellamin" si cancella la foglia di livello massimo più a destra (nell'esempio il valore 10) , se ne copia l' elemento nella radice, e si fa "scendere" tale elemento lungo un percorso radice-foglia, scambiando col minimo degli elementi contenuti nei figli.



dopo **cancellamin**

Per eseguire “inserisci” si inserisce l’elemento come foglia più a destra del livello massimo (in modo da mantenere verificate le proprietà 1 e 2) e si fa “salire” il nuovo nodo con scambi padrefiglio lungo un cammino foglia – radice fino a verificare la proprietà 3. Nel seguente esempio viene inserito l ‘elemento 4.



dopo inserimento di 4

- **Si richiede di descrivere la struttura dati astratta coda, incluse le operazioni che la caratterizzano**

La coda è un insieme dinamico in cui gli elementi sono inseriti ad un’estremità e sono estratti dall’altra. La coda implementa quindi lo schema FIFO.

La definizione del tipo di dato astratto coda è:
Dati: sequenza Q di n elementi generici
Operazioni:
Enqueue: Aggiungi x come ultimo elemento
Dequeue: Toglie da Q il primo elemento e lo restituisce

Pila

- **Descrivere la struttura dati Pila.**

La pila è una struttura dati di tipo LIFO (Last In First Out), in cui tutte le operazioni di inserimento e cancellazione avvengono alla stessa estremità (detta top). Il primo elemento che può essere tolto è quello inserito per ultimo, inoltre per accedere all'elemento i-esimo si devono togliere dalla pila tutti i successivi (non è una struttura dati ad acceso diretto).

- **Si richiede di descrivere la struttura dati astratta pila, incluse le operazione che la caratterizzano.**

La pila è un insieme dinamico in cui tutte le operazioni di inserimento e cancellazione avvengono alla stessa estremità (Detta TOP). la pila implementa lo schema LIFO (Last In First out), in cui il primo elemento che può essere tolto è quello inserito per ultimo.

La definizione del tipo di dato astratto pila è:

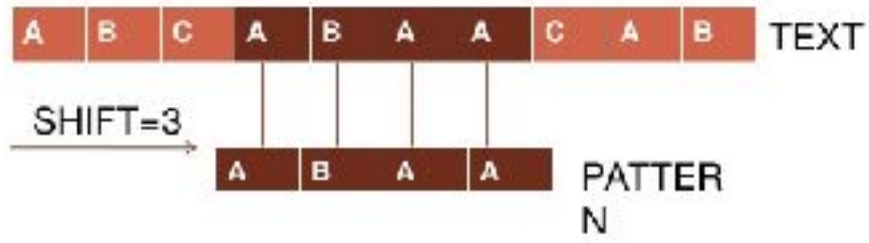
Dati: sequenza S di n elementi generici
Operazioni:
Stack Empty: restituisce true se la pila è vuota
Push: Aggiunge x come ultimo elemento
Pop: Toglie s dalla pila, ultimo elemento, e lo restituisce

String Matching

- **Dire in cosa consiste il problema dello String Matching e quale è la complessità dell'algoritmo KMP**

Il problema dello String Matching consiste nel trovare almeno una occorrenza di una sequenza P di m caratteri, detta pattern, all'interno di un'altra sequenza T di n caratteri detta testo.

Le due sequenze o stringhe P e T sono formate da caratteri tratti dallo stesso insieme A, di cardinalità finita detto "alfabeto", e sono tale che m non supera n.



L'algoritmo KMP (Knuth–Morris–Pratt) ha complessità O(n + m)

- **In che cosa consiste il problema dello String Matching approssimato? Si richiede di mostrare un esempio.**

Il problema di String Matching Approssimato consiste nel ricercare un pattern P all'interno di un testo T ammettendo errori (o differenze) tra T e P. I possibili errori sono:
i corrispondenti caratteri in P e in T sono diversi
un carattere in P non compare in T
un carattere in T non compare in P

- **Si richiede di descrivere il problema che consiste nel trovare un'occorrenza k-approssimata di un pattern P nel testo T. Mostrare un esempio di occorrenza 2-approssimata.**

Lo String Matching approssimato consiste nel cercare un pattern P all'interno di un testo T ammettendo errori tra T e P. Il problema quindi consiste nel trovare un'occorrenza k-approssimata di P in T.

k-approssimata saranno quindi il numero degli errori.

T = abcdefghi
P = dafghj

k = 2

Complessità

- **Nell'ambito della teoria della complessità dire cosa si intende per certificato polinomiale.**

Un certificato polinomiale è un algoritmo che, data una presunta soluzione del problema, verifica in tempo polinomiale che tale soluzione sia effettivamente una soluzione che dà risposta affermativa.

- **Nell'ambito della teoria della complessità, si richiede di definire le classi P e NP.**

Nell'ambito della teoria della complessità, la classe P è la classe di tutti i problemi decisionali risolvibili in tempo polinomiale con algoritmi deterministici, mentre la classe NP è la classe di tutti i problemi decisionali risolvibili in tempo polinomiale con algoritmi non deterministici.
Ovviamente la classe P è contenuta nella classe NP, poiché ogni algoritmo deterministico è anche un algoritmo non deterministico (che semplicemente non usa mai choice), ma non si sa se P sia propriamente contenuta in NP oppure se le due classi coincidono.

- **Dire cosa si intende per complessità computazione asintotica e definire la notazione asintotica "o grande" e "omega grande".**

Per poter valutare l'efficienza di un algoritmo, così da poterlo confrontare con algoritmi diversi che risolvono lo stesso problema, bisogna essere in grado di valutare l'ordine di grandezza del suo tempo di esecuzione e delle sue necessità in termini di memoria. In particolare, tale valutazione ha senso quando la dimensione dell'input è sufficientemente grande. La disciplina che si occupa della quantificazione di tali grandezze è la complessità computazione asintotica.

O (o grande) identifica il limite asintotico superiore.
Ω (omega grande) identifica il limite asintotico inferiore.
Θ (theta grande) identifica il limite asintotico stretto.

O grande è l'insieme di tutte le funzione g(n) tali che esistono due costanti positive c ed m per cui g(n) <= c f(n), per ogni n >= m.

Omega grande è l'insieme di tutte le funzione g(n) tali che esistono due costanti positive c ed m per cui g(n) >= c f(n), per ogni n >= m.

- **Si richiede di dare la definizione formale della notazione o-piccolo.**

Usata per denotare il limite superiore non asintoticamente stretto.

Divide-et-Impera

- **Quali sono le tre fasi che caratterizzano gli algoritmi di tipo divide-et-impera?**

La tecnica divide-et-impera è una tecnica usata per progettare algoritmi efficienti. Gli algoritmi che utilizzano questo metodo sono algoritmi ricorsivi. In particolare con questa tecnica, si suddivide il problema in vari sotto problemi, che sono simili al problema originale, ma di dimensioni più piccole, si risolvono i sottoproblemi in modo ricorsivo e, poi, combinano le soluzioni per creare una soluzione del problema originale. Questo metodo prevede tre passi:

- Divide: il problema viene diviso in un certo numero di sotto problemi dello stesso tipo;
- Impera: i sottoproblemi vengono risolti in modo ricorsivo (se i problemi sono piccoli si possono risolvere in maniera semplice);
- Combina: le soluzioni dei sottoproblemi vengono combinate per generare la soluzione del problema originale.

Questa tecnica è utile solo quando i sottoproblemi sono indipendenti altrimenti gli stessi sottoproblemi possono venire risolti più volte. Un algoritmo che ne fà utilizzo è il merge-sort.

Cook-Levin

- **Si richiede di enunciare il Teorema di Cook-Levin e di descrivere le sue conseguenze.**

Il teorema risponde alla domanda: “Esiste un particolare problema decisionale in NP tale che se si dimostrasse la sua appartenenza a P, allora dovrebbe risultare sicuramente P = NP?”.

Max-Heap

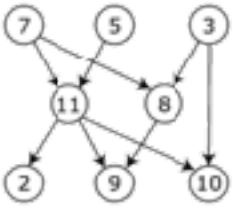
- **Cosa è un max-heap? Si richiede di fornire un esempio.**

Un Heap è una struttura dati composta da un array che possiamo considerare come un albero binario, ogni nodo dell’albero corrisponde ad un elemento dell’array. Gli heap possono essere suddivisi in Max-Heap e Min-Heap, in un max heap le chiavi/radici di ciascun nodo sono sempre maggiori o uguali di quelle dei figli, la chiave del valore massimo appartiene alla radice.

Grafo

- **Cosa si intende per ordinamento topologico di un grafo?**

Nella teoria dei grafi un ordinamento topologico è un ordinamento lineare di tutti i vertici di un grafo aciclico diretto. I nodi di un grafo si definiscono ordinati topologicamente se i nodi sono disposti in modo tale che ogni nodo viene prima di tutti i nodi collegati ai suoi archi uscenti. L’ordinamento topologico non è un ordinamento totale, poiché la soluzione può non essere unica. È possibile ordinare topologicamente un grafo se e solo se non ha circuiti.



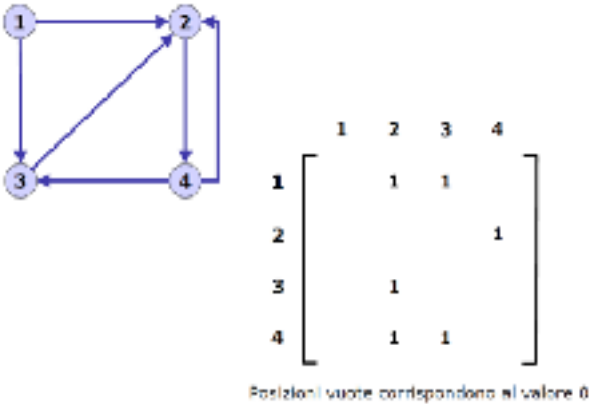
Un ordinamento topologico valido è: 7, 5, 3, 11, 8, 2, 9, 10 (graficamente da sinistra a destra e dall'alto al basso) oppure 3, 5, 7, 8, 11, 2, 9, 10 (prima i nodi con i valori minori della loro numerazione) oppure 7, 5, 11, 3, 10, 8, 9, 2 (prima i nodi con i valori maggiori della loro numerazione)

- **Cosa sono le componenti fortemente connesse di un grafo G orientato?**

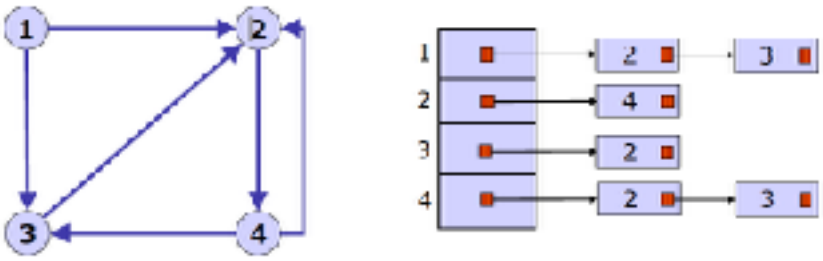
Un grafo orientato si dice fortemente connesso se per ogni coppia di nodi u e v esiste almeno un cammino da u a v ed almeno un cammino da v ad u. *(risposta da verificare)*

- **Dato un grafo G si richiede di descrivere la sua realizzazione tramite matrici e liste di adiacenza. Si richiede di mostrare un esempio per ognuna delle realizzazioni discusse e di indicare lo spazio di memoria occupato.**

Matrici di Adiacenza



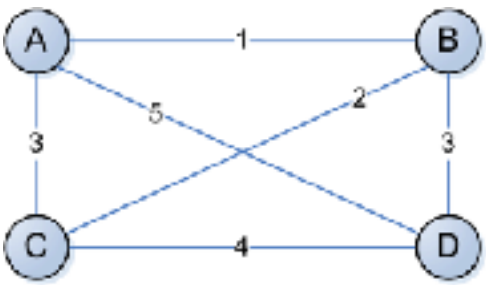
Liste di Adiacenza



Vettori di Adiacenza

Supponendo di non dover inserire ne cancellare nodi, si possono usare i vettori tramite due vettori: nodi e archi. Il vettore nodi ha n+1 posizioni, le prime per gli n nodi, l'ultima per la sentinella; nodi[i] contiene un cursore alla posizione di archi a partire dal quale è memorizzato A(i).

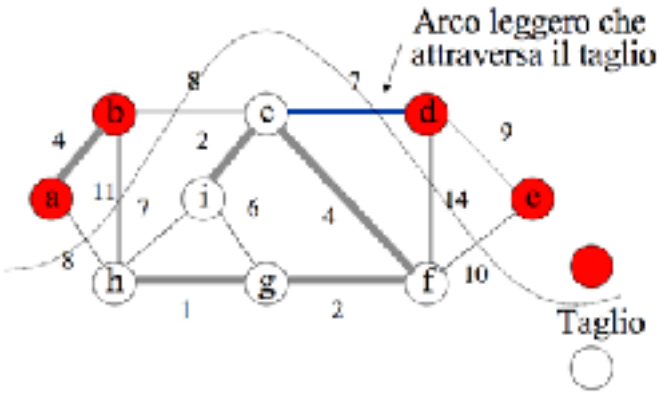
- **Dato un grafo non orientato G cosa si intende per circuito? Mostrare un esempio.**



La sequenza B-C-D-B è un circuito, la sequenza A-B-A è una catena chiusa ma non un circuito poiché è ripetuto solo un arco.

- **Dato un grafo non orientato, dare la definizione di arco leggero rispetto ad un taglio. Cosa è il taglio? Si richiede di fornire un esempio.**

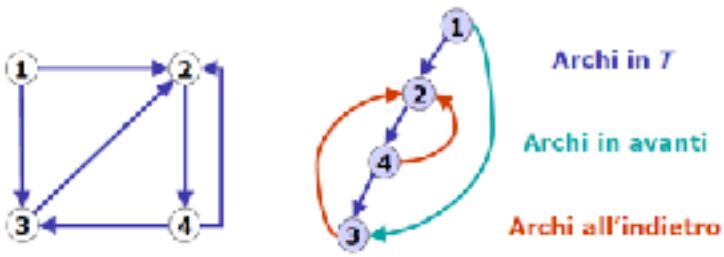
Un arco leggero è un arco con peso minimo, un taglio di un grafo non orientato G=(V,E) è una partizione di V.



- **Dato un grafo orientato G cosa si intende per albero T di copertura DFS? Quando un arco di T si definisce arco all'indietro, in avanti o di attraversamento?**

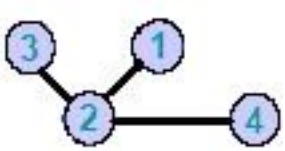
Un albero di copertura è un albero formato dagli n nodi e da n-1 archi del grafo. Un albero di copertura DFS è un albero composto dagli archi che durante una visita DFS connetto un nodo marzo ad uno non marcato; nodi ed archi formano un albero radicato T.

Se il grafo è orientato la DFS partizione gli archi in 4 sottoinsiemi:
Archi in T: archi che collegando un nodo marcato ad uno non marcato
Archi all'indietro: archi esaminati passando da un nodo di T ad un altro loro antenato
Archi in avanti: archi esaminati passando da un nodo di T ad un altro loro discendente in T
Archi di attraversamento: archi fra nodi che stanno sullo stesso livello in T



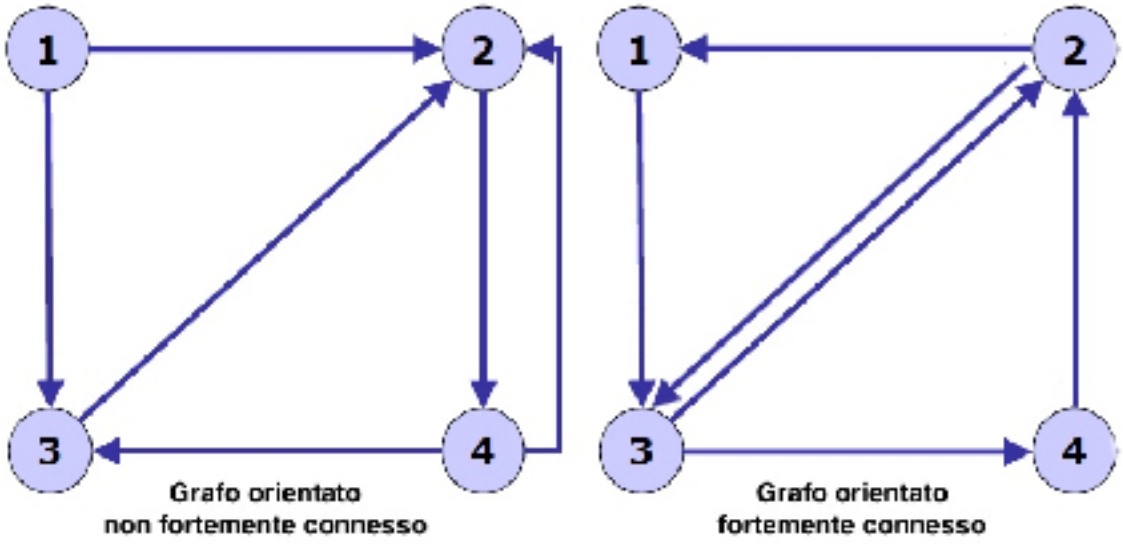
- **Definire il concetto di componente connessa in un grafo non orientato.**

Un grafo non orientato è connesso se per ogni coppia di nodi distinti u e v esiste una catena tra u e v. Un grafo non orientato è detto albero libero (albero in cui non è individuato un nodo come radice) se è connesso e per ogni coppia di nodi esiste una ed una sola catena semplice. Si può quindi affermare che un albero libero è un grafo connesso con minimo numero di archi, cioè n-1, oppure un grafo in cui non esiste un circuito.



- **Dire cosa si intende per grafo orientato fortemente connesso e mostrare un esempio.**

Un grafo orientato si dice fortemente connesso se per ogni coppia di nodi u e v esiste almeno un cammino da u a v ed almeno un cammino da v ad u.



Nel grafo non fortemente connesso esiste un cammino da 1 a 4 ma non viceversa.

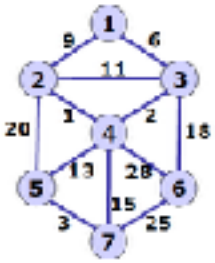
- **Dato un grafo con n nodi e m archi, si richiede di indicare in modo preciso lo spazio di memoria necessario per la rappresentazione del grafo nei seguenti casi:**
  - **A. Grafo orientato con matrice di adiacenza**
  - **B. Grafo non orientato con matrice di adiacenza**
  - **C. Grafo orientato con liste di adiacenza**
  - **D. Grafo non orientato con liste di adiacenza**

A. O(n^2)
B. O(n^2)
C. O(n+m)
D. O(n+2m)

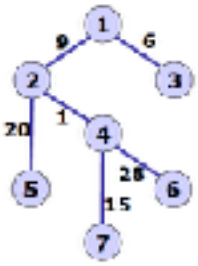
Alberto di Copertura

- **Dire cosa si intende per albero di copertura minimo e mostrare un esempio.**

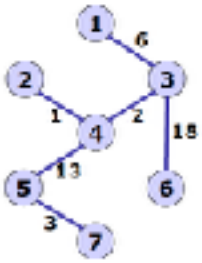
L'albero di copertura di costo minimo (o minimum spanning tree, MST) è un albero ricoprente nel quale sommando i pesi degli archi si ottiene un valore minimo.



grafo pesato



soluzione ammissibile di costo 79



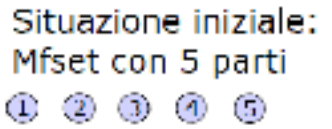
soluzione ottima di costo 43

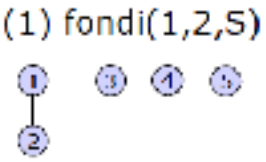
Mfset

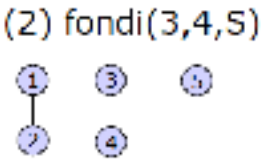
- **Definire la struttura dati Mfset e dire in cosa consiste la tecnica di compressione dei percorsi mostrando anche un esempio.**

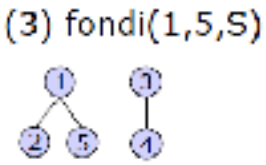
L'Mfset (Merge-Find Set) è una struttura dati derivante dal concetto di insieme delle parti, per cui dato un insieme finito di elementi a volte risulta utile partizionarli in insiemi disgiunti. L'algoritmo Mfset è quindi utile per le operazioni di Ricerca e Unione possibili su questa struttura dati.

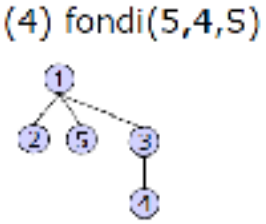
La tecnica di compressione dei percorsi consiste nel rendere figlio della radice ogni nodo che viene incontrato dalla funzione ricerca nel percorso di risalita dal generico nodo x alla radice.











Lista

- **Definire la struttura dati lista e descrivere l'implementazione basata su puntatori e cursori, evidenziando le equivalenze tra essi.**

Una lista è una struttura dati e, più esattamente, una sequenza di elementi di un certo tipo, in cui è possibile aggiungere o togliere elementi. Per fare questo occorre specificare la posizione relativa all'interno della sequenza nella quale il nuovo elemento va aggiunto o dalla quale il vecchio elemento va tolto. La lista è a dimensione variabile e ha come accesso diretto solo il primo e l'ultimo elemento mentre per gli altri va effettuata una scansione elemento per elemento.

È possibile realizzare una lista in due modi:

Realizzazione con Puntatori: Ogni elemento della lista L è un oggetto con un campo chiave Key e un puntatore Next al prossimo elemento. Esistono diversi tipi di liste: Singolarmente Concatenata (ha i campi Key e Next), Doppiamente Concatenata (campi Key, Prev e Next), Ordinata (l'elemento minimo è la testa e quello massimo in coda), Non Ordinata (elementi presentati in qualsiasi ordine), Circolare (il puntatore Prev delle testa punta alla coda, l'elemento Next della coda punta alla testa). Inoltre è possibile utilizzare una Sentinella per rendere il Delete più semplice.

Realizzazione con Cursori: È possibile realizzare puntatori con cursori, cioè non variabili interne, il cui valore è interpretato come un indice di un vettore. Questo vettore simula la memoria disponibile per i puntatori, che deve essere gestita esplicitamente nella realizzazione. Per far questo si deve definire un vettore SPAZIO che contiene tutte le liste, ognuna individuata dal proprio cursore iniziale; e contiene tutte le celle libere, organizzate anch'esse in una lista detta "listalibera". Lo SPAZIO sarà generalmente diviso in tre campi precedente, elemento e successivo.

Otteniamo quindi un equivalenza tra puntatori e cursori, ovvero p.next equivale a SPAZIO[p].next e p.prev equivale a SPAZIO[p].prev.

Greedy

- **Si richiede di definire il problema della massima diversità. Quale è la complessità dell'algoritmo Greedy più efficiente per la soluzione di tale problema? L'uso di un heap migliora la complessità (giustificare la risposta)?**

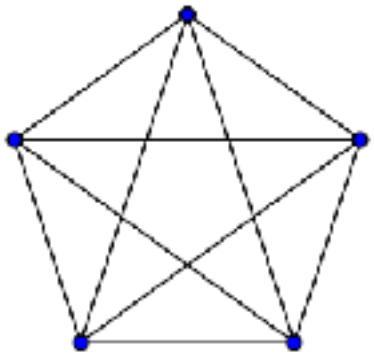
La tecnica di progettazione di algoritmi di tipo Greedy, si basa sulla semplice strategia dell' ingordo, ovvero, quella di compiere, ad ogni passo, la scelta migliore nell'immediato piuttosto che adottare una strategia a lungo termine.
Si supponga di avere monete da 11, 5 e 1 centesimo, e di dover dare un resto di 15 centesimi. L'ingordo giudicherà la moneta da 11 cent più "appetibile" delle altre, e darà un resto con una moneta da 11 e quattro da 1, utilizzando cinque monete, mentre la soluzione ottima è data da tre monete da 5 centesimi.

Il metodo greedy si può applicare a quei problemi di ottimizzazione in cui occorre selezionare un sottoinsieme S "ottimo" di oggetti.

Problema della Cricca

- **Si richiede di descrivere il problema della cricca e di mostrare un esempio con k=5.**

Una cricca è un insieme V di vertici in un grafo non orientato G, tale che, per ogni coppia di vertici in V, esiste un arco che li collega.



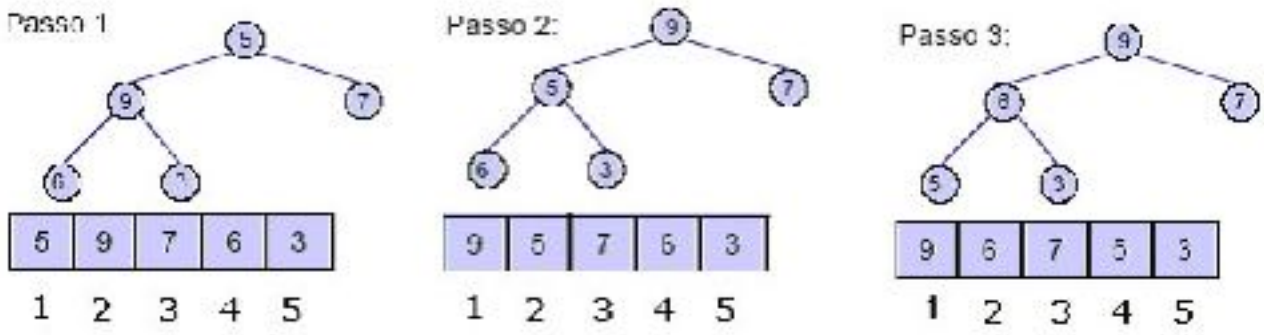
Altre Domande

- **Cosa vuol dire che un problema A si riduce in tempo polinomiale in un problema B?**

Dati A e B due problemi decisionali, si dice che A si ricuce in tempo polinomiale a B se esiste una funzione f di trasformazione tale che: f è computabile in tempo polinomiale con un algoritmo deterministico e x è un dato di ingresso per cui A ha risposta Sì se e solo se f(x) è un dato di ingresso per cui B ha risposta sì.

- **Descrivere (e mostrare un esempio di funzionamento) la procedura "restauraheap".**

La funzione restauraheap serve a sistemare quella selezione di A, che rappresenta lo heap, coinvolta dall'estrazione di un massimo.



- **Descrivere le quattro fasi che caratterizzano la progettazione di un algoritmo.**

Le fasi di progettazione di un algoritmo sono:

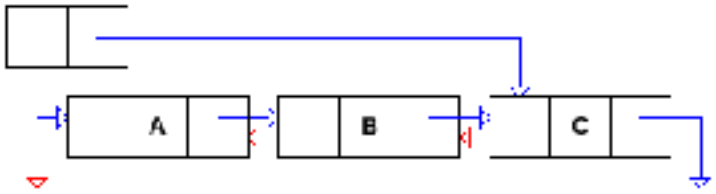
- Classificazione del problema: Si cerca di verificare l'appartenenza di un problema ad una classe più generale avente caratteristiche comuni.
- Caratterizzazione della soluzione: La caratterizzazione matematica della soluzione, quando è possibile, suggerisce algoritmi di soluzione, talvolta semplici.
- Tecnica di Progetto: Esistono delle tecnica di progetto di algoritmi che possono rendere gli algoritmi più efficienti.
- Strutture di Dati: L'impiego di opportune strutture di dati per organizzare l'input del problema può migliorare l'efficienza di un dato algoritmo.

- **Dire cosa si intende per algoritmo non deterministico e definirlo in modo formale.**

Un algoritmo non deterministico è un algoritmo che, al momento di effettuare una decisione, effettua sempre quella migliore, ossia quella che porta alla soluzione corretta.

- **Dire cosa si intende per lista bidirezionale con sentinella. A cosa serve la sentinella? Quali vantaggi offre?**

Una lista bidirezionale è anche detta lista doppiamente concatenata (campi Key, Prev e Next):



Il vantaggio derivante dall'uso di sentinelle permette di realizzare un codice più chiaro, piuttosto che più veloce. Se si usa una sentinella il codice si semplifica, ma si risparmia soltanto un tempo O(1), nelle procedure di inserimento (List-search) e di cancellazione (List-Delete).Una sentinella è un oggetto fittizio che ci consente di semplificare le condizioni di contorno a discapito di un minimo spreco di memoria (un record in più).

In altre parole, con l'uso della sentinella non mi devo andare a preoccupare se x è il primo elemento o l'ultimo presente nella lista, perché sono condizioni che non si verificano mai perché c'è sempre un elemento (o un elemento vero, oppure la sentinella). La sentinella e posta tra la testa e la coda.

- **Si richiede di descrivere la struttura dati insiemi disgiunti.**

Servono a mantenere una collezione S = {S1, S2, … Sk} di insiemi disgiunti. Ogni insieme della collezione è individuato da un rappresentate che è un particolare elemento dell'insieme.

Domande non risposte

- Data la ricorrenza T(n) = 4T(n/2) + n^3 utilizzare il metodo dell'esperto per trovare la soluzione. Si richiede di mostrare in modo dettagliato lo svolgimento.
- Data l'equazione di ricorrenza T(n) = 2T(n/2) + n log n si richiede di risolverla utilizzando il metodo dell'esperto.
- Data l'equazione di ricorrenza T(n) = 3T(n) + 2 √n si richiede di risolverla applicando il metodo dell'esperto.
- Dato un albero binario di ricerca perfettamente bilanciato (ogni nodo non foglia ha due figli) quale è la complessità dell'operazione di visita in pre-ordine dell'albero? Quale è la complessità di tale operazione quando ogni nodo non foglia ha esattamente un figlio? Si richiede di esprimere la complessità in funzione dell'altezza h dell'albero e di giustificare la risposta in modo preciso.
- Elencare le proprietà che un B-albero di ordine m deve soddisfare.
- Matrice di adiacenza e lista di adiacenza: a cosa servono? Dato un nodo, quale è la complessità computazionale di una operazione che calcola il grado del nodo con matrice di adiacenza e con le liste di adiacenza?
- Si richiede di definire la struttura dati dizionario e di descrivere come può essere realizzato.
- Si richiede di descrivere l'algoritmo di ordinamento BucketSort. Quale è la complessità nel caso migliore e peggiore?
- Si richiede di descrivere il funzionamento dell'algoritmo greedy per il calcolo di un codice di Huffman.

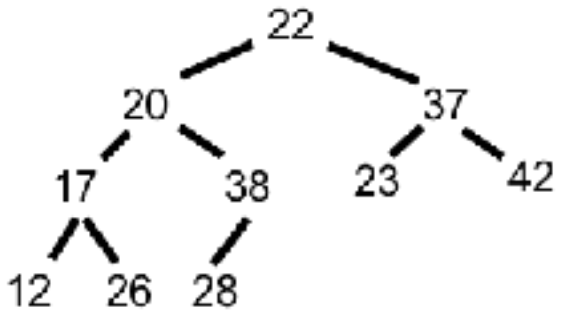
- Si richiede di specificare il costo computazionale del caso peggiore delle operazioni di DeleteMin e Insert eseguite su una coda di priorità implementata con:
    - una lista singolarmente concatenata
    - una lista ordinata singolarmente concatenata
    - un albero binario di ricerca
    - uno heap
    -

- Dato un grafo con n nodi e m archi, si richiede di indicare in modo preciso lo spazio di memoria necessario per la rappresentazione del grafo nei seguenti casi:
    - grafo orientato con matrice di adiacenza;
    - grafo non orientato con matrice di adiacenza;
    - grafo orientato con liste di adiagenza;
    - grafo non orientato con liste di adiacenza.

- Dato un B-lbero di ordine m che contiene n valori, quale è il numero di accessi a disco nel caso peggiore e migliore (usa la notazione O-grande) quando si seguono le seguenti operazioni?
    - trovare un dato valore
    - inserire un dato valore
    - cancellare un dato valore

ESERCIZI

Si consideri un albero binario completo T con 10 nodi. Etichettare i nodi con i seguenti numeri: (17,12, 37,20,28,42,23,26,22,38), in modo tale che T sia un albero binario di ricerca.



Dato il seguente algoritmo, dove A e un array e s e d sono gli indici sinistro e destro, si richiede di
valutarne la complessità computazionale (mostrare in modo dettagliato i calcoli effettuati).

```
IV(A,s,d) {
        if (s == d) {
                return false;
        }
        else {
                t = false;
                c = (s+d)/2;
                for i=s to c do {
                        for j=c+1 to d do {
                                if ((A[i]==j) and (A[j]==i))
                                        then t = true;
                        }
                }
                t1 = IV(A,s,c);
                t2 = IV(A,c+1,d);
                return (t or t1 or t2);
} }
```

T(n) = 2T(n/2) + c(n/2)^2 perché nei for non ci sono chiamate ricorsive.

Quindi per il teorema delle ricorrenza lineari con partizione bilanciata abbiamo:
alpha=log2/log2=1, Beta = 2 e quindi complessità = O(n^2)

Dato il seguente algoritmo che prende in input un intero n > 0, si richiede di determinare il costo
computazionale T(n).

```
algo(n)
        if ( n > 1 ) then
                a := 0;
                for i:=1 to n-1
                        for j:=i+1 to n do
                                a := a + 2 * (i + j);
                for i := 1 to 16 do
                        a := a + algo(n/4);
                return a;
        else
                return n-1;
```

il doppio ciclo "for" ha complessità O(n^2);
il ciclo for con ricorsione ha complessità 16T(n/4).

quindi T(n)=16T(n/4)+cn^2
applicando il metodo delle partizioni bilanciate:
alfa = log4(16)/log4(4) = 2
alfa = beta
T(n) = n^alfa*log = n^2*log(n)

Si richiede di scrivere l'equazione ricorsiva che rappresenta la complessità computazionale di p3.

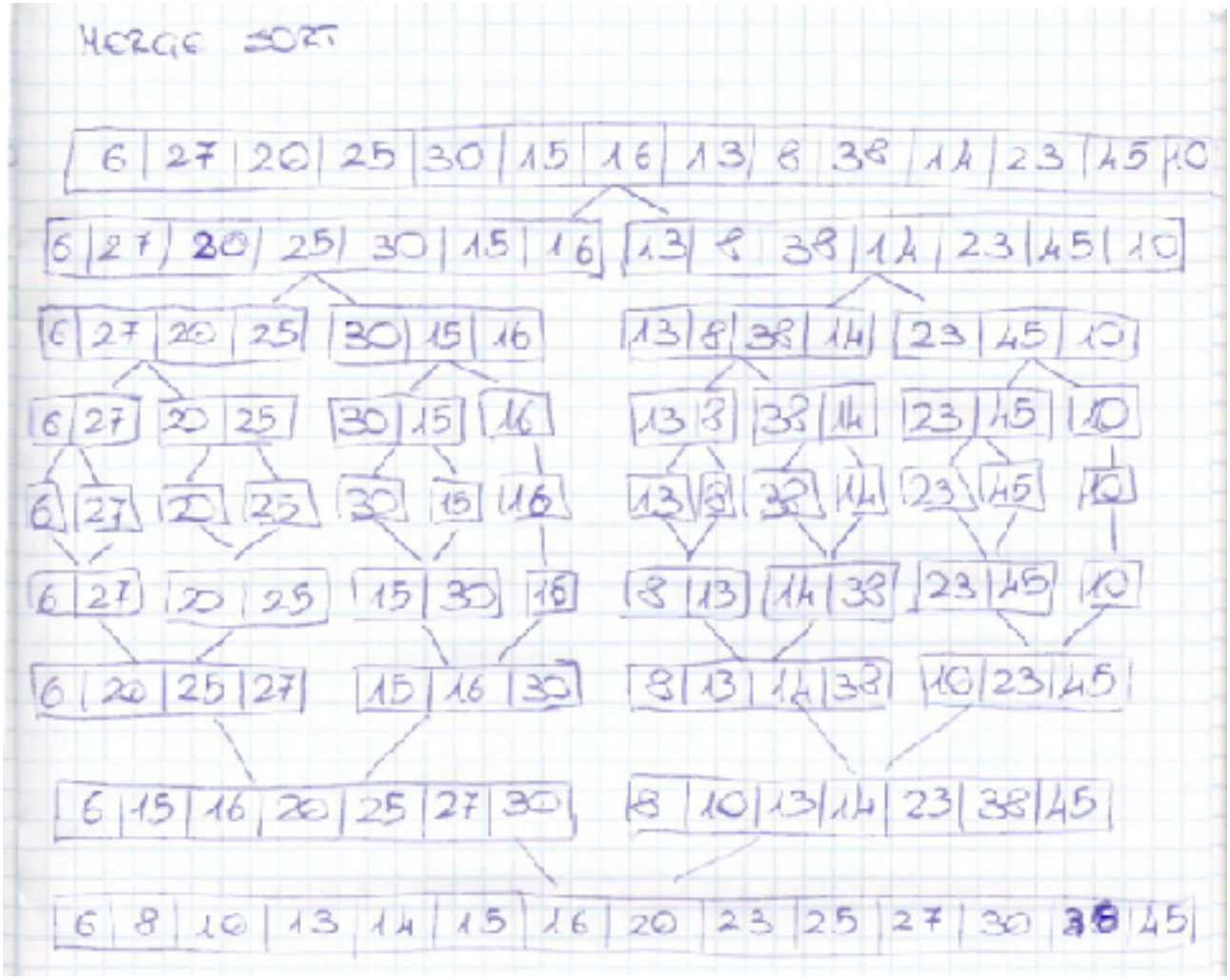
```
p3(n) {
        if n < 1
                print(n)
        for i := 1 to 9
                p3(n/3)
        p2(n)
}
p2(n) {
        if n < 1
                print(n)
        for l := 1 to 4
                p2(n/2)
        for k := 1 to n
                for j := k+1 to n
                        print(k,j)
}
```

Considerati i teoremi delle ricorrenze lineari e con partizione bilanciata che trovi in M7_U1_L1.

p2 mi risulta essere O(n^2*logn) poiché è T(n)= 4T(n/2) +cn^2 poichè ci sono 4 chiamate ricorsive (for da 1 a 4) ciascuna con n/2 più una parte di codice di complessità limitata da n^2 (i due for finali che avrebbero T=(n(n+1))/2 e quindi O(n^2))

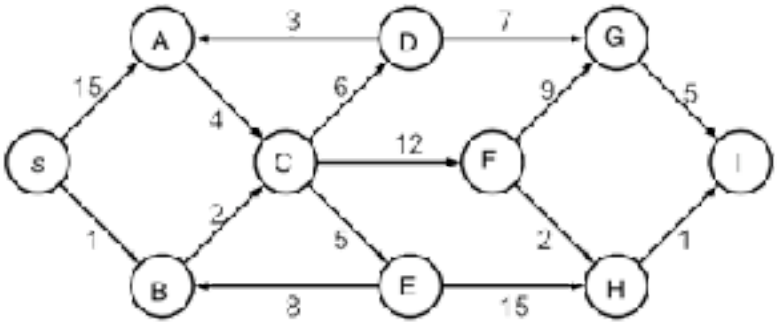
p3 è T(n)= 9T(n/3)+ cn^3 (approssimo n^2*logn con n^3) e quindi per il teorema delle partizioni bilanciate ottengo alpha = log(9)/log(3) = 2 che è inferiore a 3 e quindi p3 = O(n^3) che poi in realtà sarebbe O(n^2*logn)

Ordinare la seguente sequenza di chiavi applicando l'algoritmo Mergesort ed illustrando tutti i passi di esecuzione dell'algoritmo. Sequenza: 6, 27, 20, 25, 30, 15, 16, 13, 8, 38, 14, 23, 45, 10.

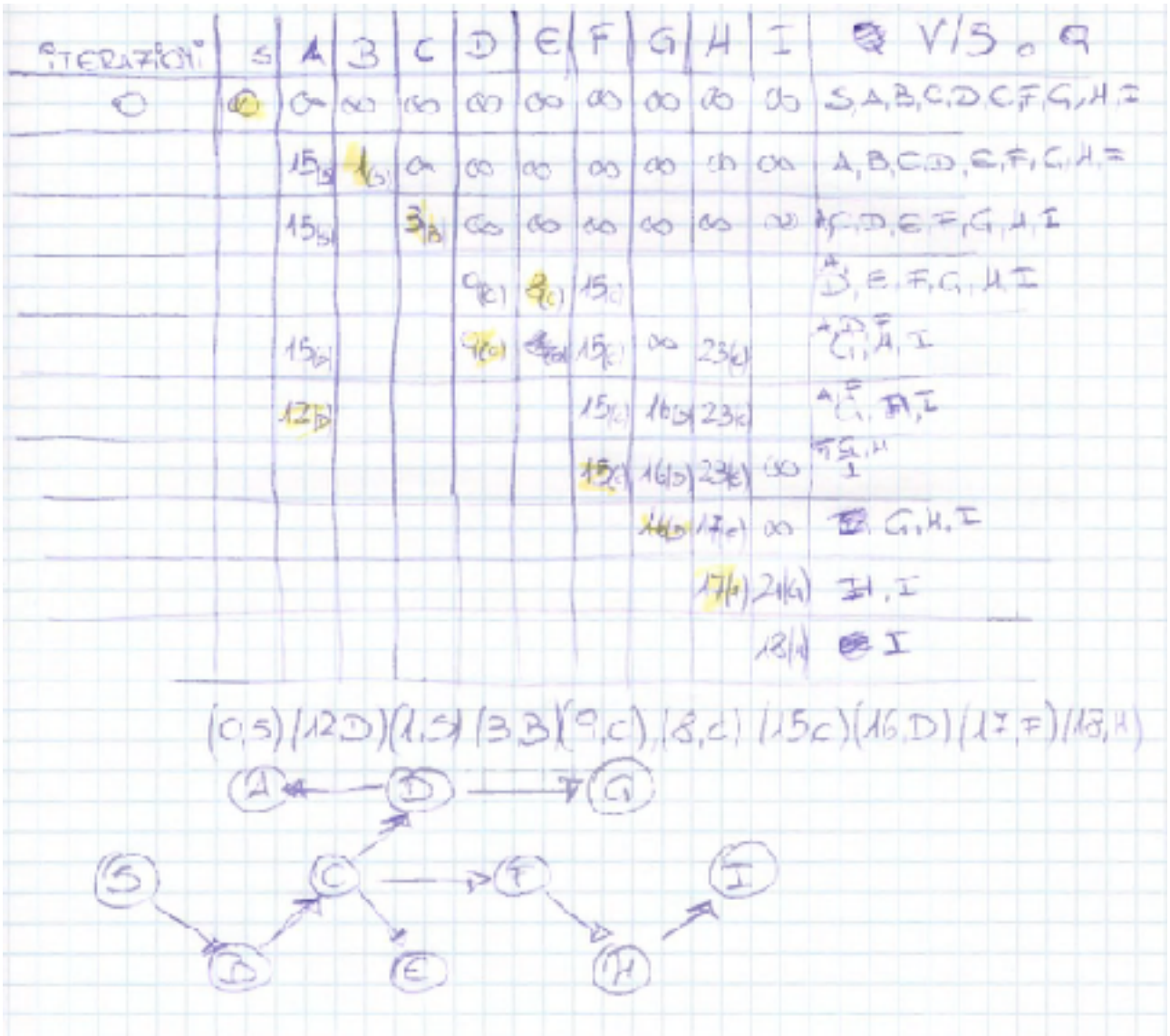


Ordinare la sequenza: 13, 10, 1, 45, 15, 12, 21, 15, 29, 34 applicando l'algoritmo Quicksort. Per ogni passo di esecuzione dell'algoritmo scegliere l'elemento più a destra come pivot ed illustrare chiaramente le partizioni trovate.

Dato il seguente grafo, si richiede di applicare l'algoritmo di Dijkstra per calcolare un albero dei cammini di costo minimo partendo dal nodo sorgente s.

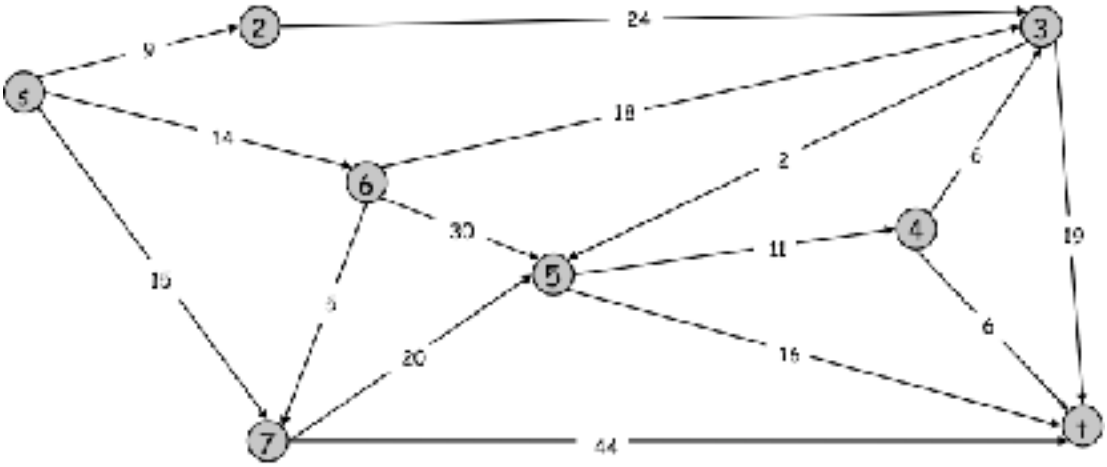


Soluzione

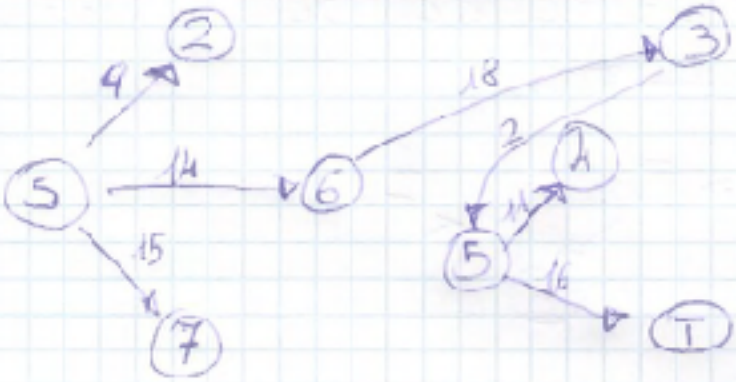


Note: La colonna V/S o Q contiene sempre i nodi del precedente passaggio non ancora selezionati, pertanto esclusi, come parte dell'albero di costo minimo.

Si richiede di applicare l'algoritmo di Dijkstra al seguente grafo partendo dal nodo S.



ITERAZIONI	S	2	3	4	5	6	7	T	Q
0	S	∞	∞	∞	∞	∞	∞	∞	S,2,3,4,5,6,7,T
		9(S)	∞	∞	∞	14(2)	15(S)	∞	2,3,4,5,6,7,T
			33(2)	∞	∞	14(6)	15(S)	∞	3,4,5,6,7,T
			32(6)	∞	44(6)		15(6)	∞	3,4,5,7,T
			32(6)	∞	35(7)			39(7)	3,4,5,T
				∞	34(3)			51(3)	4,5,T
				45(5)				50(5)	4,T
								50(4)	T
	0	9(S)	32(6)	45(5)	34(3)	14(2)	15(S)	50(4)	0



Costruire la tabella hash risultante dall'inserimento dei valori 41, 20, 50, 42, 81, 33 nell'assunzione di hashing interno con funzione primaria di hash H(k) = k mod 10, numero di celle m=10 e gestione delle collisioni tramite scansione lineare (h = 1).

La funzione di Hash è H(k) = k mod 10 pertanto si procede a rilevare il valore di H(k) per ogni valore del nostro array.

41 > H(k) = k mod 10 > 1
20 > H(k) = k mod 10 > 0
50 > H(k) = k mod 10 > 0
42 > H(k) = k mod 10 > 2
81 > H(k) = k mod 10 > 1
33 > H(k) = k mod 10 > 3

Il valore trovato corrisponderà alla posizione presunta del valore nell'array di hash, notiamo quindi che esistono collisioni in particolare per i valori 41-81, 20-50.

Scansione Lineare: H(k,i)=(H(k)+h*i) mod m.

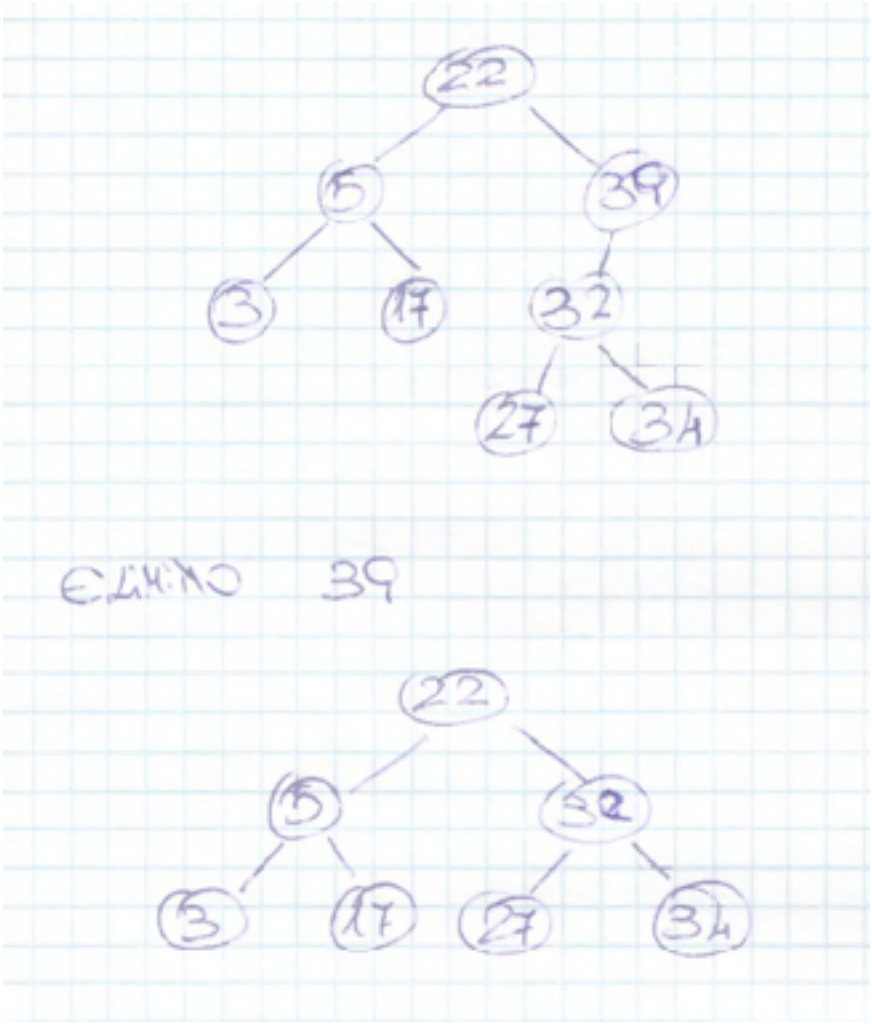
m = 10
h = 1

i = indice del numero (0 per il 41, 1 per il 20, ecc)

Collisioni:
50 > (0+1*2) mod 10 > 2 (l'indice i aumenta finché non si trova una posizione libera, i = 2)
81 > (1+1*4) mod 10 > 5 (l'indice i aumenta finché non si trova una posizione libera, i = 4)
33 > (3+1*2) mod 10 > 6 (l'indice i aumenta finché non si trova una posizione libera, i = 2)

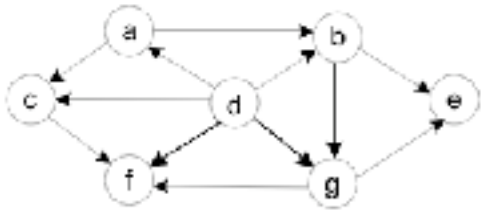
Posizione	Ordinamento
0	20
1	41
2	42
3	50
4	
5	81
6	33

Disegnare l'albero binario di ricerca risultante dopo l'inserimento delle chiavi 22, 5, 39, 32, 3, 17, 27, 34 nell'ordine indicato e mostrare l'albero dopo la rimozione della chiave 39.

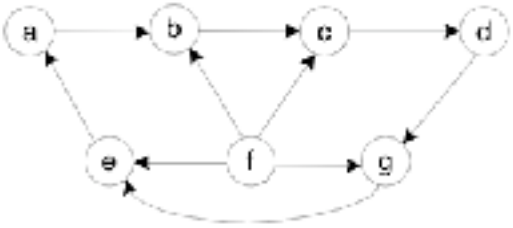


Si richiede di determinare un ordine topologico per entrambi i grafi applicando l'algoritmo basato sulla DFS. Mostrare passo passo l'esecuzione di tale algoritmo.

a.



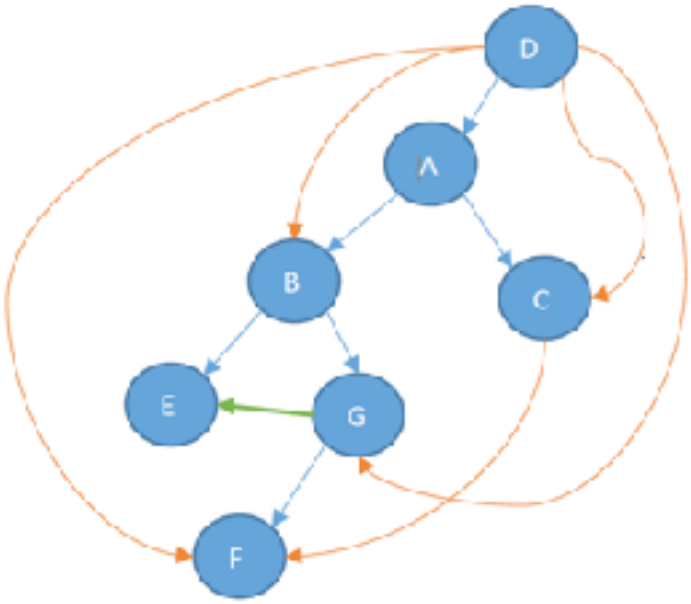
b.



Soluzione:

esercizio A

D (D,A) A (A,B) B (B,E) E (B,G) G (G,F) F (G,E) (A,C) C (C,F) (D,B) (D,C) (D,F) (D,G)



esercizio B

è presente un ciclo tra i nodi A-B-C-D-G-E, risulta quindi impossibile definire un ordinamento topologico

Dato un albero binario, i cui nodi contengono elementi interi, si scriva un algoritmo ricorsivo per ottenere l'albero inverso, ovvero un albero in cui il figlio destro (con relativo sottoalbero) è scambiato con il figlio sinistro (con relativo sottoalbero).

```
void inverti(nodo u, albero T)
{
  if(foglia(u,T))
  {
    return(0);
  }
  else
  {
    if(!sinistrovuoto(u,T) && !destrovuoto(u,T))
    {
      switch(figliodestro(u,T), figliosinistro(u,T))
      inverti(figliodestro(u,T), T)
      inverti(figliosinistro(u,T), T)
    }
    else if(!sinistrovuoto(u,T) && destrovuoto(u,T))
    {
      switch(NULL, figliosinistro(u,T))
      inverti(figliodestro(u,T), T)
    }
    else if(sinistrovuoto(u,T) && !destrovuoto(u,T))
    {
      switch(figliodestro(u,T), NULL)
      inverti(figliosinistro(u,T), T)
    }
  }
}
```

Scrivere un algoritmo con complessità O(n) che prende in input un numero reale x ed un array ordinato A che contiene n numeri reali e determinare se in S esistono due numeri reali la cui somma è uguale a x.

```
int A[n];
int x;
i=primolista(A);
j=ultimolista(A);

for(while i<j)
{
        if(i+j<x) succlista(i);
        else if(i+j>x) preclista(j);
                else /*trovato*/ break;
}
```

Dato il seguente algoritmo, dove M e una matrice di n righe ed m colonne, si richiede di valutarne la complessità computazionale (mostrare in modo dettagliato i calcoli effettuati).

```
Algo (M)
{
        crea un array A di lunghezza nXm
        for i=1 to n
                for j=1 to m
                        A[(i-1)*m+j] = M[i][j]
        MERGE-SORT(A)
        for i=1 to n
                for j=1 to m
                        M[i][j] = A[(i-1)*m]+j]
}
```

il primo ciclo for ha dimensione n*m;
il merge-sort come da definizione ha dimensione nmlg(nm);
il secondo ciclo for ha dimensione n*m;

la complessità è data quindi dalla procedura di merge-sort, cioè I(nmlg(nm))

Dato il seguente pseudocodice, si richiede di calcolarne la complessità computazionale. Giustificare la risposta.

```
best(1,n)
max=A[1] ;d=1; s=1
for i=1..n do
sum = 0
for j=i..n do
sum = sum + A[j]
if (sum > max)
then
{max = sum; d = i; s = j;}
return (max,d,s)
```

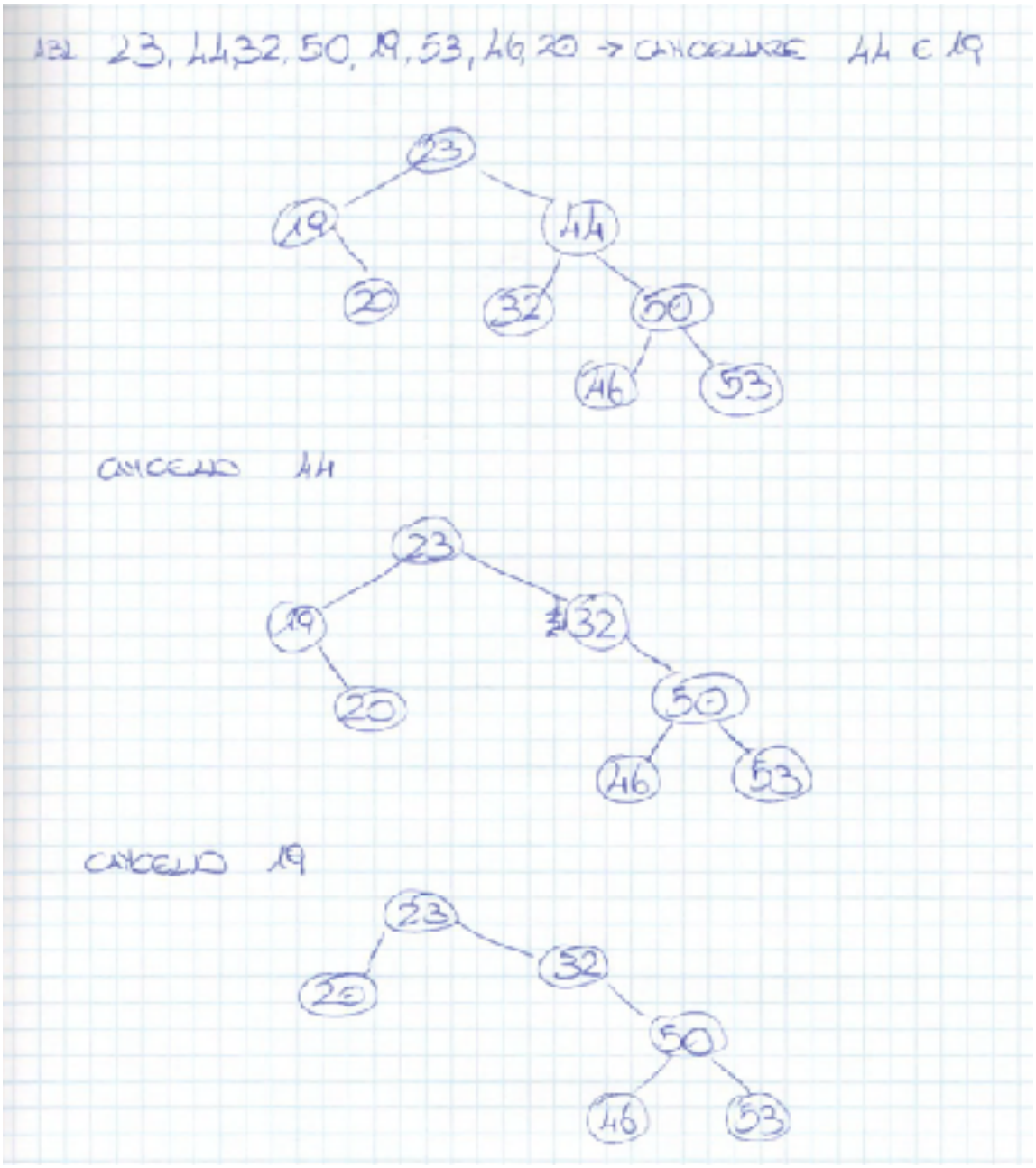
La complessità nel caso pessimo è data dal ciclo for inserito nell'altro ciclo for, quindi è O(n^2)

Si richiede di calcolare la complessità computazionale della seguente funzione.

```
algo(n) /* n: numero intero */
{
        if (n <= 2) then return n ;
        else return algo(n-1) - 2*algo(n-2);
}
```

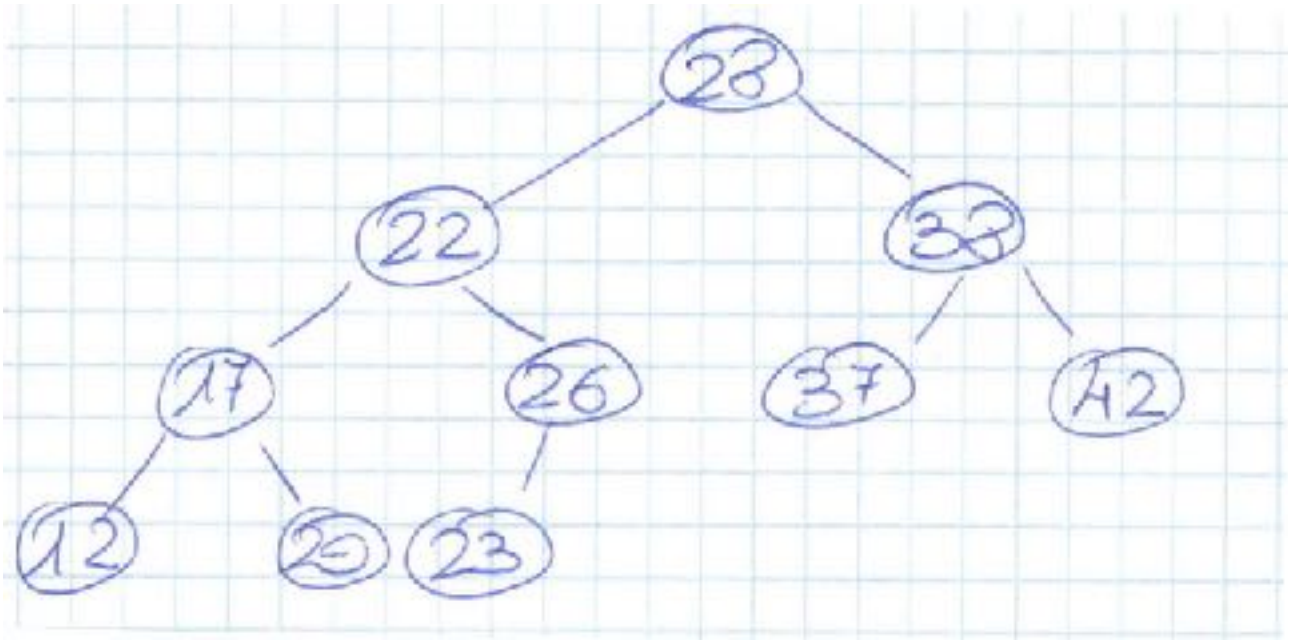
La complessità dello pseudocodice è O(2^n)

Inserire in un albero binario di ricerca inizialmente vuoto le chiavi 23,44,32,50,19,53,46,20 (nell'ordine dato) illustrando l'albero finale. Cancellare poi nell'ordine le chiavi 44 e 19 illustrando l'albero dopo ogni cancellazione.

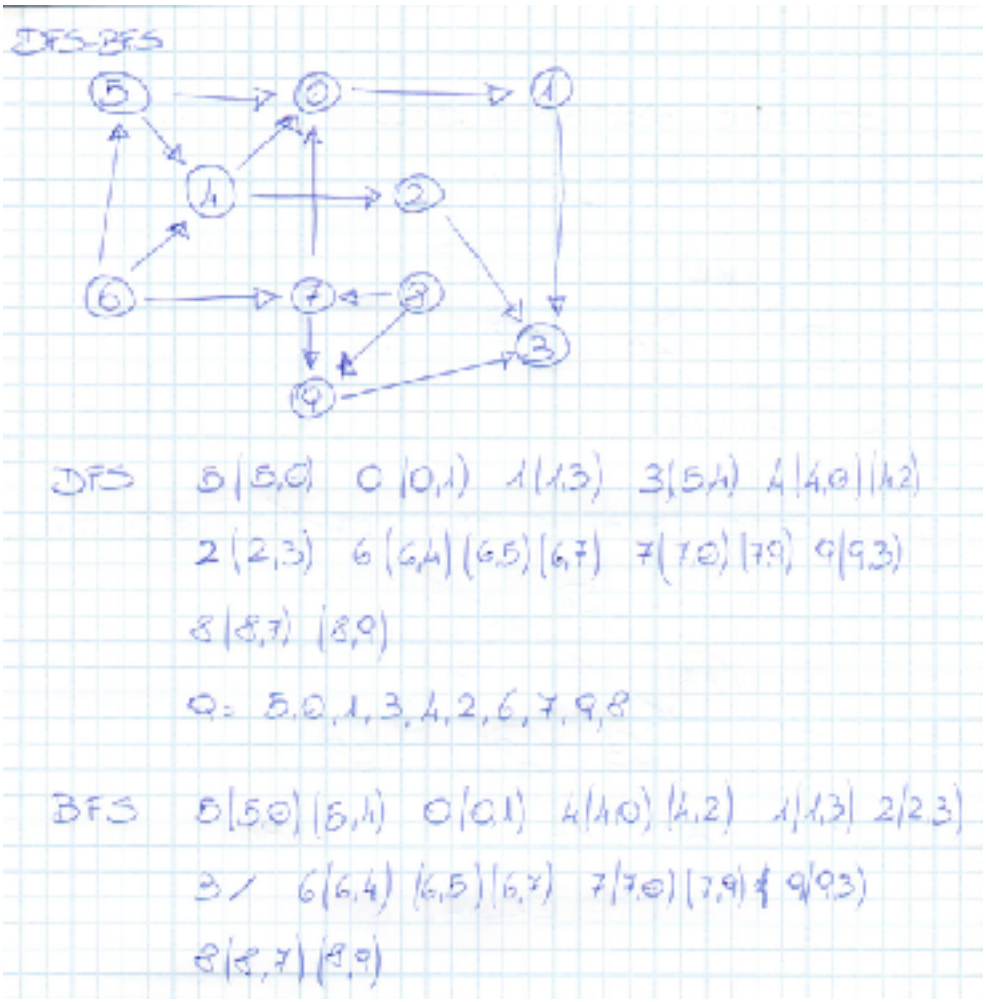


"Si consideri un albero binario completo T con 10 nodi, etichettare i nodi con i seguenti numeri: 17,12,37,20,28,42,23,26,22,38 in modo tale che T sia un albero binario di ricerca.

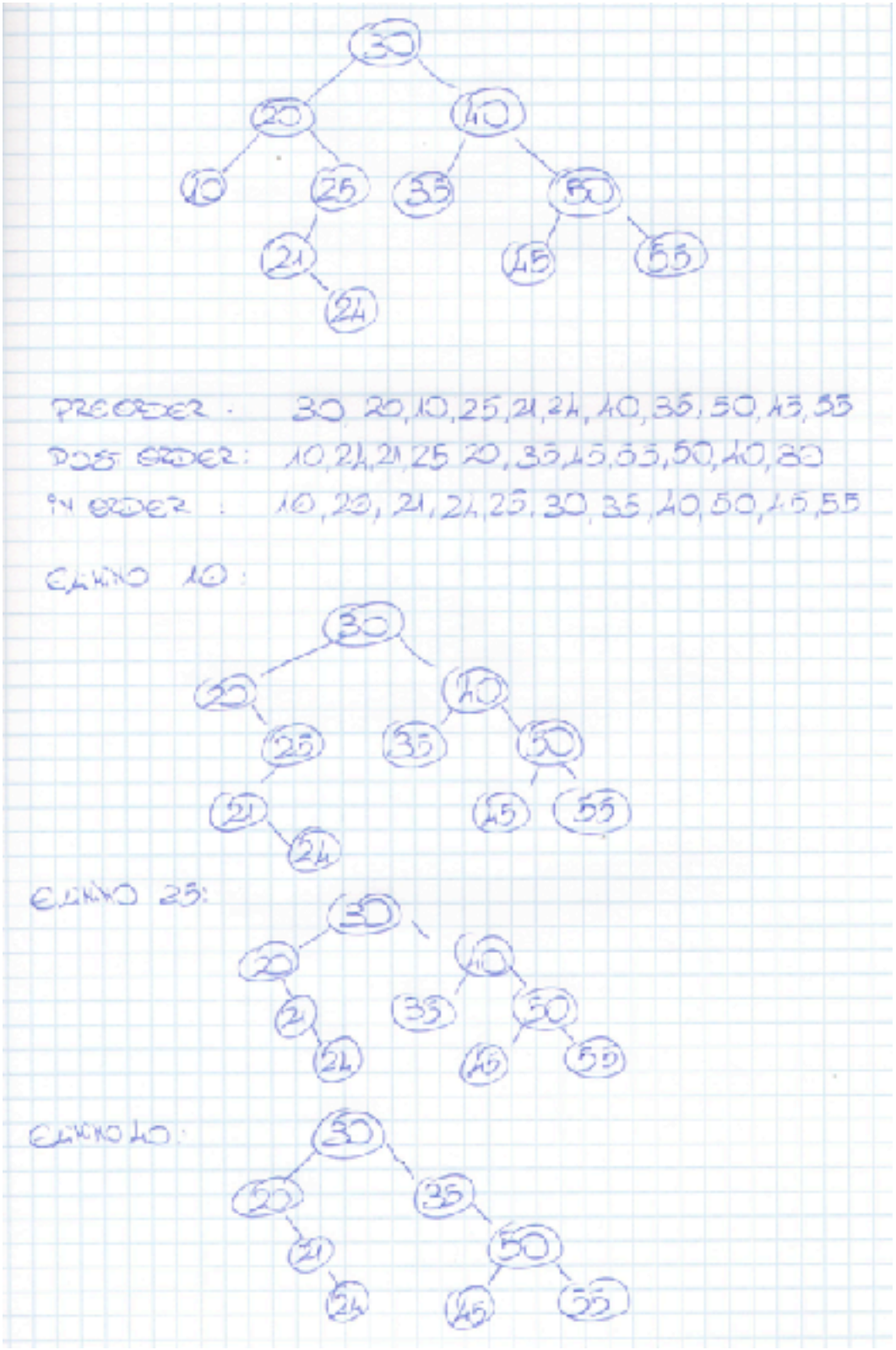
Essendo richiesto che sia un albero binario completo si deve rispettare la regola: "Un albero binario completo è un albero binario in cui ogni livello, fino al penultimo, è completamente riempito. L'ultimo livello è riempito da sinistra a destra"



BFS e DFS


DFS-BFS
DFS 5|5,0| 0|0,1| 1|1,3| 3|5,4| 4|4,0||4,2|
2|2,3| 6|6,4||6,5||6,7| 7|7,0||7,9| 9|9,3|
8|8,7||8,9|
Q: 5,0,1,3,4,2,6,7,9,8
BFS 5|5,0||5,1| 0|0,1| 4|4,0||4,2| 1|1,3| 2|2,3|
3/ 6|6,4||6,5||6,7| 7|7,0||7,9| 9|9,3|
8|8,7||8,9|

Si richiede di costruire un albero binario di ricerca inserendo successivamente i seguenti valori: 30, 40, 50, 20, 35, 10, 25, 45, 21, 55, 24. In particolare si richiede di:
1) disegnare l'albero dopo ogni inserimento;
2 indicare quindi in che ordine vengono stampati i nodi dell'albero quando si utilizzano i metodi di visita pre-order, post-order ed in-order;
3) eliminare i valori 10, 25 e 40, disegnando l'albero risultante dopo ogni cancellazione e descrivendo come viene effettuata ognuna di esse.



Inserire le chiavi 10, 22, 31, 4, 15, 28, 17, 88, 59 in una tabella di hash di lunghezza m = 11 utilizzando la funzione hash h(k) = k mod m. Illustrate il risultato dell'inserimento di queste chiavi utilizzando la scansione lineare ed il doppio hashing con h'(k) = 1 + (k mod (m - 1)).

Scansione Lineare

Posizione	0	1	2	3	4	5	6	7	8	9	10
Valore	22	88			4	15	28	17	59	31	10

- 10 mod 11 = 10
- 22 mod 11 = 0
- 31 mod 11 = 9
- 4 mod 11 = 4
- 15 mod 11 = 4

Posizione 4 già occupata sfrutto quindi la funzione della Scansione Lineare

H(k,i) = (H(k) + h * i) mod m > (4 + 1 * 1) mod 11 > 5 mod 11 > 5

h: non è indicato, li attribuisco il valore 1.
i: è un indice progressivo aumenta dal valore 1 finché non si trova una posizione libera.

- 28 mod 11 = 6
- 17 mod 11 = 6, posizione già occupata ripeto quindi la Scansione Lineare trovando:

(H(k) + h * i) mod m > (6 + 1 * 1) mod 11 > 6 mod 11 > 6 > collide
aumento l'indice i:
(H(k) + h * i) mod m > (4 + 1 * 2) mod 11 > 5 mod 11 > 7

- 88 mod 11 = 0, posizione già occupata ripeto quindi la Scansione Lineare trovando:

(H(k) + h * i) mod m > (0 + 1 * 1) mod 11 > 1 mod 11 > 1

- 59 mod 11 = 4, posizione già occupata ripeto quindi la Scansione Lineare trovando:

(H(k) + h * i) mod m > (4 + 1 * 1) mod 11 > 5 mod 11 > 5
(H(k) + h * i) mod m > (4 + 1 * 2) mod 11 > 6 mod 11 > 6
(H(k) + h * i) mod m > (4 + 1 * 3) mod 11 > 7 mod 11 > 7
(H(k) + h * i) mod m > (4 + 1 * 4) mod 11 > 8 mod 11 > 8

Doppio Hashing

Posizione	0	1	2	3	4	5	6	7	8	9	10
Valore	22		59	17	4	15	28	88		31	10

- 10 mod 11 = 10
- 22 mod 11 = 0
- 31 mod 11 = 9
- 4 mod 11 = 4
- 15 mod 11 = 4

Posizione 4 già occupata sfrutto quindi la funzione del Hashing Doppio

H(k,i) = (H(k) + i * H'(k)) mod m > (H(k) + i * (1 + (k mod (m - 1)))) mod m

(4 + 1 * (1+ (15 mod (11-1))) mod 11 = 4+6 mod 11 = 10

Posizione 10 già occupata, aumento l'indice:

(4 + 2 * (1+ (15 mod (11-1))) mod 11 = 4+12 mod 11 = 5

- 28 mod 11 = 6
- 17 mod 11 = 6 , collisione:

(6 + 1 * (1+ (17 mod (11-1))) mod 11 = 3

- 88 mod 11 = 0, collisione

(0 + 1 * (1+ (8 mod (11-1))) mod 11 = 9
(0 + 2 * (1+ (8 mod (11-1))) mod 11 = 7

- 59 mod 11 = 4, collisione

(4 + 1 * (1+ (59 mod (11-1))) mod 11 = 3

(4 + 2 * (1+ (59 mod (11-1))) mod 11 = 2