Domande Orale di APA con risposte

Algoritmi e programmazione (Politecnico di Torino)

DOMANDE CAMURATI:

DISCLAIMER:
Questo file contiene le domande di Camurati fatte all'orale con relative risposte.
Sono redatte da me in quanto studente, non sono quindi fonte di verità assolute e, alcune cose, potrebbero essere non proprio corrette.
→ Non sono responsabile di eventuali fallimenti universitari. ←
Usatelo come vi pare e piace, ma non come arma.
(E sì, lo faccio per la gloria)
KEEP SPREADING

→ Heap e sua complessità

L'heap è una particolare struttura dati ad albero binario con 2 particolari proprietà:

→ la prima proprietà è detta "strutturale" e implica che l'albero binario sia (quasi) completo ovvero che siano completi tutti i livelli tranne eventualmente l'utlimo (l'albero si riempie da sinistra verso destra per livelli.
→ la seconda proprietà è detta "funzionale" e implica che la chiave contenuta nella radice dell'albero sia maggiore rispetto alle chiavi contenute nel figlio sinistro e nel figlio destro.

L'heap viene inoltre implementato non tramite puntatori ma tramite vettore.

La funzione HEAPify() è una funzione ricorsiva che trasforma in heap un albero binario, ovvero permette all'albero binario di assumere la proprietà funzionale. (ovvero permette di avere nella radice la chiave maggiore tra radice, figlio left e figlio right). Ha complessità O(log n);

La funzione HEAPbuild() trasforma un intero albero binario memorizzato in un vettore in un heap, facendogli acquistare la proprietà funzionale → applica la heapify a partire dal padre dell'ultima foglia fino alla radice. La complessità è O(n);

La funzione HEAPsort() permette di estrarre in sequenza ordinata le chiavi presenti all'interno dell'heap, è in loco ma non stabile. Ha complessità O(nlog n);

Se utilizzato per implemtare una coda a priorità PQ, l'heap permette inserimenti e estrazioni con complessità logaritmica O(logn).

→ Formula di Stirling

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n, \quad \text{per } n \to +\infty$$

Permette di approssimare un fattoriale.

→ BST, confronto BST con Heap, confrontare le altezze

Entrambi BST che Heap sono alberi binari, ma sono diverse le proprietà che li caratterizzano: Mentre un heap deve mantenere una certa proprietà strutturale per poter essere definito tale (ovvero deve essere quasi completo con inserimento dei nodi da destra a sinistra per livelli), il BST non ha una limitazione di questo genere e può degenerare facilmente in una lista concatenata portando ad una sua degradazione delle prestazioni (da complessità logaritmiche a complessità lineari).

Inoltre l'heap deve mantenere una proprietà funzionale, ovvero deve avere nella radice una chiave che sia maggiore di quelle contenute rispettivamente nell'albero sinistro e nel sottoalbero destro. Il BST possiede una proprietà simile ma comunque diversa in quanto prevede che nel sottoalbero sinistro sia prevista la chiave minore di tutta la terna di chiavi e che nel sottoalbero destro sia prevista la chiave maggiore delle tre. In questo modo, avendo la chiave della radice, possiamo dire che nel sottoalbero sinistro saranno contenute solamente chiavi minori della radice mentre nel sottoalbero destro saranno contenute solo chiavi maggiori della radice.
Poiché il BST non è vincolato da una proprietà strutturale, possiamo avere una grande discrepanza tra la crescita per livellli dei due alberi binari → il bst potrà crescere nel caso peggiore come una lista linkata mentre l'heap sarà costretto a crescere per livelli nodo dopo nodo da sinistra verso destra.

→ formula del numero di archi in un grafo completo e significato e funzione di albero ricoprente minimo.

Un grafo completo è un grafo che possiede il numero massimo di archi possibili, ovvero ogni vertice è collegato ad un altro vertice da un arco diretto: Nel caso di un grafo NON orientato abbiamo un numero di archi pari a [|V| * (|V| - 1)] / 2. → Questo risultato si ricava dal fatto che il numero di archi è pari alle combinazioni di |V| elementi presi due a due.
Nel caso di un grafo orientato invece abbiamo un numero di archi pari a [|V| * (|V| - 1)]. (Ovvero occorre contare il doppio numero di archi perché un arco sarà l'andata e l'altro sarà il ritorno).

Un albero ricoprente minimo è un albero ricavabile da un grafo pesato che connette la totalità dei vertici del grafo nel quale sommando i pesi degli archi si ottiene un valore minimo. Un albero ricoprente minimo è ricavabile tramite gli algoritmi di Prim e/o l'lagoritmo di Kruskal.

→ complessità inferiore di un algoritmo di ordinamento basato sul confronto, dimostrazione

La complessità minima ottenibile con un algoritmo di ordinamento basato sul confronto è O(nlogn) (come per MergeSort e QuickSort) e si dimostra riportando l'esito di un singolo confronto tra due numeri in un labero delle decisioni. Poiché in un confronto ci sono due possibili alternative (minore o maggiore), tale albero è anche un albero binario.
Per esempio se gli elementi del vettore da ordinare sono tre, si ha un massimo di 3! = 6 (permutazioni semplici) ordinamenti possibili (corrispondenti alle soluzioni foglia).
Generalizzando possiamo dire che nel caso di n elementi si ha un numero pari a n! possibili ordinamenti e tali ordinamenti corrispondono al numero di foglie dell'albero.
L'albero ha un'altezza (ovvero la distanza foglie radice massima) pari al numero di confronti, poiché l'albero è binario, il numero della foglie è pari a 2^h.
Poiché le soluzioni sono n! allora 2^h deve essere necessariamente maggiore.
Da qui, tramite la formula di Stirling si può giungere all'approssimazione 2^h >= n! > (n / e)^n e ricavando h si giunge a h > nlogn. SI ricava così il limite inferiore alla complessità di un algoritmo di ordinamento basato sul confronto.

→ calcolo del numero di nodi di un bst (scritto in maniera ricorsiva ed in C)

```
void pre_order_visit(link root, int *count) {
        if (h==NULL)
                return;
        printf("%s", root→val);
        pre_order_visit(root→left, *count + 1);
        pre_order_visir(root→right, *count + 1);
}
```

→ calcolo del minimo RICORSIVAMENTE

```
int MinSearch(link root) {
        if (root==NULL) return root→val;
        if (root→left == NULL) return root→val;
        MinSearch(root→left);
}
```

→ deduzione da qui della sua equazione alle ricorrenze e risoluzione di questa.

{ Analisi di complessità di un albero completo: problema di tipo “divide and conquer”
a = 2 (2 sottoproblemi alla volta) e b = 2 (divido i dati a metà ogni volta).
Poiché non conosciamo a priori la dimensione iniziale dell’albero, possiamo supporre che, dati n nodi totali e n – 1 nodi totali senza radice, la dimensione dei sottoproblemi sia ogni volta di (n – 1) / 2. Approssimo conservativamente a n / 2 ottenendo T(n) = 1 + aT(n/b) ovvero T(n) = 1 + 2T(n/2) con T(1) = 1. Di conseguenza, svolgendo l’equazione alle ricorrenze, ottengo una complessità lineare O(n). }
Nel caso della ricerca del minimo in un BST la complessità sarà logaritmica nel caso di un albero bilanciato.

→ differenza tra Dijkstra e visita in ampiezza

Una visita in ampiezza permette, in un grafo orientato e non pesato, di determinare tutti i vertici raggiungibili da un determinato vertice ‘s’ detto ‘sorgente’, calcolando la distanza minima da tra ‘s’ e tutti i i vertici da esso raggiungibili.

Dijkstra permette invece, in caso di grafi pesati, di trovare un cammino minimo tra un vertice sorgente e tutti gli altri vertici. Un cammino minimo è quel percorso che collega due vertici e che minimizza la somma dei costi associati all’attraversamento di ciascun arco.

→ cosa si intende per componente fortemente connessa (in che tipo di grafi e cosa vuol dire "massimale") e qual è il limite inferiore alla complessità degli algoritmi di ordinamento con relativa dimostrazione (approssimazione di Stirling, ecc.)

Componente connessa: sottografo (non orientato) connesso massimale (ovvero il più grande possibile per cui la proprietà di mutua raggiungibilità è valida)
Componente fortemente connessa: sottografo (orientato) fortemente connesso massimale

Una componente fortemente connessa è un sottografo orientato, quindi un sottoinsieme di archi orientati e un sottoinsieme di vertici del grafo originale che rispettano una proprietà di mutua raggiungibilità. Una componente fortemente connessa massimale è una componente connessa alla quale non è possibile annettere vertici adiacenti, massimale è diversa da massima poiché possono coesistere più componenti massimali in un grafo, ma tra queste sono una è massima, ovvero ha cardinalità maggiore di tutte le altre.

→ cos'è un albero ricoprente minimo e su quale tipo di grafo si applica

Un albero ricoprente minimo è un albero ricavato da un grafo pesato che comprende tutto l’insieme dei vertici del grafo ma un sottoinsieme degli archi, se il grafo ricoprente è minimo questo implica che gli la somma totale dei pesi degli archi è minima.
Si applica appunto sui grafi pesati.

→ definire e spiegare l'equazione delle ricorrenze

L’equazione alle ricorrenze permette un’analisi di complessità di una funzione ricorsiva sul numero dei dati ‘n’ in ingresso. T(n) viene espressa tramite 3 termini:
D(n): costo necessario per la fase di divisione
C(n): costo necessario per la fase di ricombinazione
Costo della soluzione elementare: supponiamo sia unitaria
a = numero di sottoproblemi che risulta nella fase di Divide
b = fattore di riduzione, quindi n / b è la dimensione di ciascun problema

Si ottiene quindi $T(n) = D(n) + aT(n/b) + C(n)$ → quando siamo sopra una certa soglia e il problema non è il caso terminale
$T(n) = O(1)$ → per problemi banali (casi terminali)

→ cosa vuol dire che i sottoproblemi sono indipendenti.

I sottoproblemi, nel divide et impera, sono da considerarsi indipendenti ppoiché la risoluzione di un particolare sottoproblema non viene riutilizzata nel caso tale sottoproblemi si ripresenti in tempi futuri. Per ovviare a questo problema esistono nuovi paradigmi come la programmazione dinamica o la memoization.

→ cos'è il backtrack

Nei programmi ricorsivi il backtrack è uno ‘smarcamento’, ovvero la sottrazione di un determinato valore dall’insieme della soluzione (ancora incompleta).
Nelle funzioni ricorsive viste si ha backtrack quando per esempio si giunge nella condizione terminale, si testa la correttezza della soluzione attuale ritrovata e, nel caso che questo test fallisca, si fa backtrack sull’ultima soluzione inserita nell’insieme per poter vagliare le altre scelte disponibili per quel livello di ricorsione.

→ cosa sono gli algoritmi greedy

Il paradigma Greedy permette, in alcuni casi, di arrivare ad una soluzione ottima senza necessariamente esplorare tutto lo spazio delle soluzioni come invece viene fatto nel paradigma divide et impera. Si ottiene quindi in questi casi una soluzione in tempi minori, algoritmi Greedy ottimi sono per esempio l’algoritmo di Dijkstra dei cammini minimi o l’algoritmo di scheduling.

→ tabelle di hash, linear probing e double hashing

Le tabelle di Hash sono particolari tabelle di simboli che abbattono la complessità per quanto riguarda inserimenti e ricerche. Per gli inserimenti infatti si ottengono complessità O(1), così come per le ricerche. Vengono implementate tramite vettori e utilizzano funzioni di Hash, che trasformano la chiave di ricerca in un indice della tabella.
Poiché generalmente il numero di elementi da inserire è minore della dimensione della tabella stessa, la funzione di Hash può restituire un indice per una determinata chiave uguale a un indice assegnao ad un’altra chiave, questo fenomeno si chiama collisione.
Per ovviare alla collisione, esistono vari modi tra cui il linear probing e il double hashing.

Il primo prevede, in caso di collisione di una chiave con un altra già in tabella, di incrementare l’indice della chiave da inserire. Questo incremento viene eseguito fino a che non si trova una

locazione libera per la chiave. Se, a forza di incrementi si arriva a superare la dimensione della tabella, allora si ricomincia a scansionare la tabella dall'inizio.

Il Double Hashing invece prevede, in caso di collisione, l'utilizzo di una seconda funzione di hash, che operi sul risultato ottenuto dalla prima, questo ragionamento viene iterato finché non viene trovata una postazione libera per la chiave.

→ tipologie di funzioni di hash in base ai tipi di dato (metodo modulare, moltiplicativo,…)

Esistono varie funzioni di Hash, che si adattano a seconda del dato da inserire nella tabella.
In caso di chiavi in virgola mobile appartenenti ad un intervallo prefissato si considera il 'metodo moltiplicativo', che consiste nel normalizzare la chiave all'intervallo, in modo da ottenere un indice compreso tra 0 e M – 1.

```
int hash (float k, int M) {
return ((k – s) / (t- s) )  * M;
```

Per chiavi di tipo intero si utilizza in questo caso il 'metodo modulare', ovvero prendere la chiave intera, dividerla per la dimensione della tabella ottenendo il resto della divisione e utilizzandolo come indice per la chiave. (Occorre avere M che non sia potenza della base)
Per chiavi di tipo intero si puù anche utilizzare un metodo 'moltiplicativo-modulare, dove prima di eseguire il modulo con la dimensione della tabella, si esegue una moltiplicazione cella chiave con il real A = 0,618033.

Per chiavi di tipo stringa si utilizza un 'metodo modulare' in cui, considerandola carattere per carattere, si moltiplica ciascun carattere per la base della codifica ASCII (128). Così si ha oerò un enorme consumo di risorse per l'elevamento a potenza della base, rischiando inoltre di portare l'indice in overflow molto rapidamente.
Con le stringhe si preferisce allora utilizzare il metodo di valutazione a la 'Horner' in cui, inoltre, ad ogni passo viene eseguito il modulo per la dimensione della tabella del risultato ottenuto, evitando overflow dell'indice.

→ algoritmo di ordinamento stabile/in loco

Un algoritmo di ordinamento può essere identificato come stabile o non stabile e in loco o non in loco.
Un metodo di ordinamento si dice stabile se preserva l'ordine relativo dei dati con chiavi uguali all'interno del file da ordinare. Ad esempio se si ordina per anno di corso una lista di studenti già ordinata alfabeticamente un metodo stabile produce una lista in cui gli alunni dello stesso anno sono ancora in ordine alfabetico mentre un ordinamento instabile probabilmente produrrà una lista senza più alcuna traccia del precedente ordinamento.

Un algoritmo in loco utilizza locazioni di memoria ausiliare in numero fisso, ovvero la memoria occupata è indipendente dal numero di dati, oltre ai dati stessi).

→ counting sort

Il counting sort è un particolare algoritmo di ordinamento interno basato sul conteggio (complessità O(n)) ed è stabile ma non in loco. Esso opera solo su chiavi intere, quindi nel caso lo si volesse utilizzare con chiavi non intere occorre utilizzare allo stesso tempo una tabella di simboli.
Esso utilizza oltre al vettore di dati in input altri due vettori C e B utilizzati ripettivamente per il calcolo delle occorrenze semplici / multiple e vettore temporaneo di fine elaborazione. (B infatti, a fine algoritmo verrà copiato in A).
L'algoritmo alloca il vettore B di pari dimensione del vettore A in input, mentre alloca il vettore C pari al valore della chiave massima presente in A + 1.
Si calcolano dapprima le occorrenze semplici, ovvero quante chiavi dello stesso valore sono presenti nel vettore A. L'informazione viene memorizzata nel vettore C. → C[A[i]] ++;
Si calcolano poi le occorrenze multiple, calcolando, per ogni chiave, quanti chiavi (contando anche i duplicati) la precedono. → C[i] += C[i -1];
Si procede poi a ritroso nel vettore A, inserendo in ordine le chiavi a seconda delle occorrenze multiple presenti in C. Al termine si avrà nel vettore B tutte le chiavi del vettore A ma in corretto ordine. Al termine si prosegue con la copiatura di B in A.

Dal punta di vista pratico, il numero della chiave maggiore presente in A e il numero totale delle chiavi totali deve essere all'incirca dello stesso ordine. Questo perché, nel caso contrario, si dovrebbe allocare il vettore C per molte posizioni in più di quante effettivamente utilizzate, sprecando molta memoria con possibili difficoltà di allocazione dello stesso.

→ heap sort

L'heap sort è un algoritmo di ordinamento ricorsivo utilizzato sulla struttura dati heap.
Esso scambia nel vettore dell'heap la prima chiave con l'ultima, decrementa poi la dimensione della tabella e applica una Heapify a partire dalla radice per poter ristabilire la proprietà funzionale dell'heap. Ripete questa procedura fino allo svuotamento dell'heap, ottenendo un ordinamento nelle chiavi estratte.
Ha complessità O(n log n) ed è in loco ma non stabile.

→ Cammino di Eulero

Dato un grafo non orientato e 2 suoi vertici, si dice Cammino di Eulero un percorso che connette tali vertici attraversando ogni arco una e una sola volta. (E' l'opposto di un cammino di Hamilton, basta scambiare il ruolo di vertici e archi)

→ Definizione multigrafo

Un multigrafo è un grafo in cui sono previsti archi multipli che connettono la stessa coppia di vertici, e si può generare se in fase di inserimento degli archi non si scartano quelli già esistenti (esempio dei ponti di Kneiphof)

→ Dato un albero con V nodi quanti archi ha?
Un albero deve avere V – 1 nodi poiché deve mantenre la proprietà dell'albero, ovvero deve essere aciclico. Se infatti avesse un arco in più allora saremmo in presenza di un ciclo e di conseguenza di un grafo.

→ Cosa succede se gli aggiungo/tolgo un arco
Risposta precedente

→ Definizione DAG
Un DAG è un grafo diretto aciclico, dove per diretto si intende orientato.

→ Come si verifica se sono presenti cicli in un grafo?
Attraverso una visita in profondità si scansionano tutti i vertici e se costruisce una foresta di alberi della ricorsione (o un singolo albero se il grafo è connesso). Passando poi all'etichettatura degli archi, se esistono uno o più archi di tipo Backward allora siamo in presenza di un ciclo.

→ Definizione ricorsiva albero binario

Ricorsivamente, un albero binario può essere visto come:
. un insieme di nodi vuoto (condizione di terminazione)
. una radice, un sottoalbero sinistro e un sottoalbero destro.

→ Cos'è un heap e come si implementa

Un heap è una struttura dati ad albero binario implementata non tramite puntatori bensì tramite vettore.
Esso si attiene a due particolari proprietà: una proprietà strutturale (l'albero deve essere completo o quasi completo, in quanto l'inserimento di dati avviene per livelli e da sinistra verso destra) e una proprietà funzionale (l'albero deve avere nella radice una chiave che sia maggiore di quelle contenute sia nel sottoalbero sinistro che nel sottoalbero destro).

Viene appunto implementato tramite vettore dove, supponendo che la radice si trovi all'indice 'i', allora il figlio sinistro si trova all'indice (i * 2) mentre il figlio destro si trova all'indice (i * 2) +1.

→ Cosa vuol dire albero quasi completo

Un albero quasi completo è un albero a cui manca un nodo e tale nodo è l'ultimo figlio a destra.

→ Cos'è un punto di articolazione

In un grafo un punto di articolazione è un particolare vertice che, se eliminato, disconnette in due o più parti il grafo stesso. L'osservazione di eventuali punti di articolazione viene fatta sull'albero della visita in profondità generatosi dal grafo.
Un punto di articolazione può essere la radice dell'albero della visita in profondità se questo prevede il diramarsi di due o più sottoalberi. Può essere inoltre un nodo intermedio se esiste almeno un sottoalbero di visita in profondità ivi radicato che non presenti alcun arco backward che punta a un antenato proprio del vertice.
Le foglie dell'albero della visita in profondità non possono essere punti di articolazione.

→ Cos'è un grafo connesso

Un grafo si dice connesso se è possibile raggiungere ogni vertice a partire da un determinato vertice. Ovvero se, presi due vertici qualunque, esiste un cammino che li connette.

→ DAG: definizione e algoritmi

Un DAG è un grafo diretto (orientato) e aciclico. Esso viene utilizzato in particolare per schematizzare un gruppo di tasks.
Attraverso una visita in profondità è possibile determinare un ordinamento topologico inverso (nel caso i nodi vengano cosiderati per tempi di completamento crescenti) oppure secondo un ordinamento topologico (nel caso i nodi vengano considerati secondo tempi di completamento decrescenti).

→ cos'è un arco sicuro

Un arco sicuro definisce un arco pesato utilizzato negli algoritmi di Prim e Kruskal per la ricerca di un albero ricoprente minimo. Un arco sicuro è un arco che attraversa un taglio e tale arco è quello a peso minimo tra tutti gli archi che attraversano tale taglio.
Inoltre un arco è sicuro se e solo se aggiunto ad un sottoinsieme di un albero ricoprente minimo

→ cos'è un taglio

Un taglio è una partizione sui vertici di un grafo

→ cos'è un ciclo di Hamilton e a quale tipo di classe appartiene

Un ciclo di Hamilton è un cammino semplice che va da vertice sorgente e si richiude sullo stesso vertice sorgente e che tocca ogni vertice di un grafo una e una sola volta.
Determinare ricorsivamente se esiste un cammino semplice che collega il nodo ‘s’ al nodo ‘d’.
Ritorno con successo se il cammino è lungo |V| - 1.
Setto la cella dell’array visited per marcare i nodi già visitati, reset quando ritorno con insuccesso (backtrack).

La complessità è esponenziale.

→ cos'è la classe NP-completa

E’ una classe formata da problemi NP, tali che, se si trova un algoritmo polinomiale per un problema di questa classe, allora attraverso opportune trasformazioni polinomiali si possono trovare algoritmi polinomiali per tutti i problemi NP.

→ BST: definizione ricorsiva

Un albero binario di ricerca può essere visto ricorsivamente come:
. un insieme di nodi vuoti (condizione di terminazione)
. una radice con un sottoalbero sinistro e un sottoalbero destro, per costruzione la radice contiene un valore della chiave maggiore del sottoalbero sinistro e minore del sottoalbero destro.

→ rotazione a SX e definire la struct associata ad un vertice V

```
link rotL(link h) {
        link x = h→r;
        h→r = x→l;
        x→l = h;
        return x;
}
```

→ Quando Dijkstra fornisce soluzione ottima

Dijkstra, algoritmo per il calcolo dei cammini minimi, non fornisce una soluzione ottima quando siamo n presenza di archi a peso negativo e non funziona proprio quando siamo in presenza di cicli a peso negativo.

→ Quando Bellman-Ford restituisce soluzione ottima e come stabilisce se la soluzione non è corretta

Bellmann-ford restituisce una soluzione ottima anche quando abbiamo un arco a peso negativo, e rileva la presenza di un ciclo negativo se dopo |V| - 1 iterate ne facciamo un'altra in cui ha effetto la relaxation degli archi.

→ Definizione di componente fortemente connessa

Una componente fortemente connessa è un sottografo orientato, quindi un sottoinsieme di archi e un sottoinsieme di vertici che presenta la proprietà di muta raggiungibilità per tutti i vertici che la compongono.

→ Complessità del quicksort nel caso peggiore e scrivere la T(n) relativa a questo caso

Il caso peggiore del quicksort coincide con il caso in cui il vettore dato in input all'algoritmo sia già ordinato, in questo caso la complessità è quadratica. Spesso non si considera il caso peggiore del quicksort perché è facilmente evitabile utilizzato un flag che indice nel caso se la struttura dati è già ordinata o meno. Per il resto, nel caso medio esso è linearitmico.
Nel caso peggiore la T(n) = T(n – 1) + n → O(n^2).

→ algoritmo ricorsivo per calcolo prodotto di due numeri di n cifre

n deve essere pari a 2^k.
Se la dimensione è 1, allora la condizione è di terminazione e calcolo il semplice prodotto tra due numeri.
Se n > 1:
Divido il numero x in 2;
Divido il numero y in 2:
Ottengo 4 sottoproblemi e calcolo xs * ys || xs * yd || xd * ys || xd * yd.

→ calcolo t(n) algoritmi ricorsivi, ed eccezioni es quicksort caso peggiore.

La T(n) è una funzione che ci permette di esprimere la complessità di algoritmi ricorsivi.
T(n) = D(n) + aT(n / b) +C(n)
dove 'a' è il numero di sottoproblemi che risulta dalla fase di divide
dove 'b' è il fattore di riduzione, quindi n/ b è la dimensione di ciascun sottoproblema

→ Complessità visita ricorsiva in un BST nel caso migliore e in quello peggiore

→ Ricerca ricorsiva del massimo in un BST.

```
Item maxR (link root) {
if (root == NULL) return NULLItem;
if (root→ right == NULL) return root→ val;

return maxR(root →right);
}
```

→ Ricerca ricorsiva del massimo in un albero binario qualsiasi (senza le proprietà di BST).

```
void maxR(link root, int *max) {
```

```
if (root→left == NULL && root→right == NULL) {
        if (root→ val > *max)
                *max = root→ val;
}
maxR(root→left, max);
maxR(root→right, max);
}
```

→ Come dev'essere strutturato un BST per poter trovare il k-esimo elemento, e come avviene la ricerca.

La normale struttura del BST dev'essere estesa con un campo indicante il numero di nodi radicato nel nodo corrente.
Rango k = chiave in posizione k nell'ordinamento, se k = 0 è la chiave minima.
T è il numero di nodi nel sottoalbero sinistro, se:

t == k: ho trovato la chiave con rango k, ritorno la radice del sottoalbero
t > k: ricorro nel sottoalbero sinistro alla ricerca della k-esima chiave più piccola.
t < k: Ricorsione nel sottoalbero destro alla ricerca della chiave k -t -1 esima

→ la codifica prefix-free e codici di huffman

La codifica prefix-free implica che nessuna parola di codice valida sia anche un prefisso di un'altra parola di codice, ovvero non può essere una sottostringa di un'altra parola di codice.
Per l'operazione di codifica abbiamo una giustapposizione di stringhe mentre per la decodifica dobbiamo percorrere un albero binario.
I codici di Huffman permettono di costruire l'albero binario per i codici prefissi a lunghezza variabile. Essendo a lunghezza variabile questi codici permettonon di risparmiare memoria e assegnare una codifica in base alla frequenza con cui viene trovato un certo simbolo.

Nei codici di Huffmann troviamo una coda a priorità organizzata per frequenze crescenti, l'algoritmo è greedy.
Vengono estratti i 2 simboli (o aggregati) a mino frequenza.
Si sommano le frequenze dei due simboli e si inserisce in foglia il risultato.
Si inserisce la radice dell'albero nella coda a priorità.
Si termina quando la coda è vuota.

Poiché l'albero è binario e le operazioni di estrazione e inserimento sono fatte in una coda a priorità la complessità è linearitmica.

→ cosa si intende per altezza di un albero

L'altezza di un albero binario è la distanza massima tra i nodi foglia e la radice dell'albero.
Ovvero la profondità massima.

→ risolvere un sort ricorsivo e relativa equazione alle ricorrenze

quick sort $T(n) = 2T(n / 2) + n$

Merge sort uguale

→ Funzione di hash da applicare alle stringhe

Si può utilizzare un metodo modulare per stringhe possibilmente corte:
Le stringa viene divisa carattere per carattere e ciascuno di essi viene valutato in base alla tabella ASCII e moltiplicato per la base ASCII (128). (Di questa base se ne fa una potenza, poiché la vautazione è polinomiale) → metodo alla Horner
M è un numero primo maggiore della dimensione della tabella

```
int hash (char *nome, int M)                    →elimina a ogni passo i
                                                        multipli di M
{
        int h = 0, base = 127;
        for (; *v != '\0'; v++)
                h = (base * h + *v) % M;
return h;
}
```

Questo metodo consuma però una quantità enorme di risorse per l'elevamento a potenza, e si rischia di generare interi talmente gradi da andare in overflow.

→ open addressing, differenze tra linear probing e double hashing

Nell'open addressing, ogni cella della tabella di simboli può contenere al massimo un indice alla volta e di conseguenza la dimensione della tabella deve essere tale per poter memorizzare tutti gli elementi.

La tecnica del linear probing adotta una funzione di hash per poter trovare la posizione in tabella associata a quella determinata chiave, se tale posizione è già occupata siamo in caso di conflitto. In questi casi si incrementa di uno l'indice della tabella dove inserire la chiave fino a che non si trova una posizione libera. Se infatti la posizione è nuovamente occupata, si incrementa nuovamente l'indice della tabella. Se viene scandita la tabella fino alla sua terminazione, allora si ricomincia dall'indice iniziale della stessa. La funzione di liner probing provoca un elevato effetto di clustering, dove tutte le chaivi inserite di seguito formano un cluster, un blocco, di celle piene che occorrerà scandire una per una finoa trovare la fine di questo blocco e poter effettuare un corretto inserimento.

Il double hashing prevede invece l'utilizzo di due funzioni di hash, la prima basata sul numero primo più prossimo alla dimensione della tabella, mentre la seconda funzione adotta un numero primo maggiore. Nel caso di collisione con il primo tentativo, si procede con il calcolo della funzione come: (k mod m + 1+ k mod M) mod m dove k mod m è la prima funzione di hash mentre 1 + k mod M + la seconda funzione di hash, la loro somma mod m restituisce l'indice.

→ Tabelle di simboli, in generale (inevitabilmente poi si è finito a parlare di hash)

Le tabelle di simboli sono particolari strutture dati che permettono di associare a dati di tipo non intero un corrispettivo intero. Permettono quindi due operazioni essenziali come inserimento e ricerca.
Tra le varie tabelle di simboli possiamo trovare le tabelle ad accesso diretto (che sprecano molta memoria e per questo non utilizzate);

Gli array non ordinati (con complessità nella ricerca lineare e inserimento unitario);
Gli array ordinati (che permettono inserimento lineare e ricerca binaria logaritmica);
Le lista non ordinate (che permettono inserimenti unitari e ricerche lineari)
Le liste ordinate (che permettono inserimenti e ricerche lineari)
Gli alberi binari di ricerca, con operazioni logaritmiche se l'albero è bilanciato e operazioni lineari se l'albero degenera in una lista linkata;
Le tabelle di hash, che permettono inserimenti unitari e ricerche unitarie nel caso medio.

→ La relaxation e relativo lemma

La relaxation è un'operazione utilizzata negli algoritmi dei cammini minimi come Dijkstra e Bellman-ford: Essa è il calcolo della stima della distanza di un determinato vertice del grafo dal vertice utilizzato come sorgente. Nell'algoritmo di Dijkstra la relaxation viene applicata una sola volta e permette di stimare una distanza inizialmente infinita (poiché inizialmente viene fatta una stima conservativa) fino a convergere alla vera e propria distanza.

Lemma: il sottocammino di un cammino minimo è anch'esso un cammino minimo.

→ Tecnica del left child right sibling

Un albero può essere rappresentato tramite la rappresentazione left child right sibling.
Nella struttura dati che compone l'albero di grado k (il grado è il massimo numero di nodi pendenti che posso trovare in un albero) si ha infatti un puntatore al padre, una chiave, un puntatore al fratello a destra e un puntatore al figlio sinistro.
La rappresentazione è efficiente infatti si hanno sempre 3 puntatori, indipendentemente dal grado della'albero.
In questa rappresentazione il padre non conosce tutti i figli ma solo il primogenito tra essi, che conosce il secondogenito e così via.

→ Definizione di Grafo Completo.

Un grafo si dice completo se tutti i vertici presenti sono connessi agli altri vertici tramite un arco diretto. In questo modo il grafo è completo se possiede il numero massimo di archi possibili. Questo impone, se |V| è la cardinalità dei vertici, un numero di archi pari a |V| * (|V| - 1) / 2 se il grafo è orientato mentre |V| * (|V| - 1) se il grafo non è orientato.

→ Componente fortemente connessa, componente connessa, cammino di Hamilton

Una componente fortemente connessa è un sottografo orientato, quindi un sottoinsieme di archi e un sottoinsieme di vertici che presenta la proprietà di mutua raggiungibilità per tutti i vertici che la compongono.
Una componente connessa invece è un sottografo non orientato, quindi un sottoinsieme di archi non orientati e un sottoinsieme di vertici che presenta la proprietà di mutua raggiungibilità per tutti i vertici che la compongono.
Un cammino di Hamilton è un cammino che collega un vertice di partenza s a un vertice di destinazione d e che attraversa tutti i vertici che compongono il grafo una e una sola volta.

→ Rappresentazione dei grafi e loro complessita spaziali

Un grafo può essere rappresentato mediante 2 strutture dati: la lista delle adaicenze o la matrice delle adiacenze.
La prima corrisponde a un vettore di liste di interi, (nel caso in cui i vertici non siano interi si sfrutta una tabella di simboli), che, per ogni cella, specifica con quali altri nodi è connesso il vertice corrispondente a quella cella.
Nella matrice della adiacenze invece si sfrutta una vera e propria matrice di interi, dove per ogni cella viene inserito un numero che indica la presenza o meno di un arco e questo numero può corrispondere al peso dell'arco se l'arco è pesato. La matrice delle adiacenze risulta simmetrica se il grafo non è orientato mentre non simmetrica se il grafo è orientato.

→ formula degli archi in un grafo completo (orientato e non orientato)

Un grafo si dice completo se tutti i vertici presenti sono connessi agli altri vertici tramite un arco diretto. In questo modo il grafo è completo se possiede il numero massimo di archi possibili. Questo impone, se |V| è la cardinalità dei vertici, un numero di archi pari a |V| * (|V| - 1) / 2 se il grafo è orientato mentre |V| * (|V| - 1) se il grafo non è orientato.

→ Codici Prefix-free

La codifica prefix-free implica che nessuna parola di codice valida sia anche un prefisso di un'altra parola di codice, ovvero non può essere una sottostringa di un'altra parola di codice.

→ Dimostrazione limite inferiore ordinamento basato sul confronto

La complessità minima ottenibile con un algoritmo di ordinamento basato sul confronto è O(nlogn) (come per MergeSort e QuickSort) e si dimostra riportando l'esito di un singolo confronto tra due numeri in un albero delle decisioni. Poiché in un confronto ci sono due possibili alternative (minore o maggiore), tale albero è anche un albero binario.
Per esempio se gli elementi del vettore da ordinare sono tre, si ha un massimo di 3! = 6 (permutazioni semplici) ordinamenti possibili (corrispondenti alle soluzioni foglia).
Generalizzando possiamo dire che nel caso di n elementi si ha un numero pari a n! possibili ordinamenti e tali ordinamenti corrispondono al numero di foglie dell'albero.
L'albero ha un'altezza (ovvero la distanza foglie radice massima) pari al numero di confronti, poiché l'albero è binario, il numero della foglie è pari a 2^h.
Poiché le soluzioni sono n! allora 2^h deve essere necessariamente maggiore.
Da qui, tramite la formula di Stirling si può giungere all'approssimazione 2^h >= n! > (n / e)^n
e ricavando h si giunge a h > nlogn. SI ricava così il limite inferiore alla complessità di un algoritmo di ordinamento basato sul confronto.

→ Complessità algoritmo delle torri di Hanoi e ricavare la T(n) relativa. Definizione di grafo completo.

La complessità dell'algoritmo delle torri di Hanoi è esponenziale (più preciso è O(2^n)) e si determina dall'equazioni delle ricorrenze T(n) = 2T(n – 1) + 1
dove 'a' = 2 perché si risolve 2 sottoproblemi di dimensione n – 1 ciascuno.
E' un algoritmo di divide and conquer.

→ Quick union, quick find

Quick union e Quick find fanno parte di un gruppo di 3 algoritmi riguardanti il problema della Online Connectivity. Questo problema consiste nel determianare quali coppie inserite in input

rappresentato connessioni ignote in precedenza o non implicate transitivamente dalle precedenti, senza esplicitamente costruire il grafo.
Nella quick union gli insiemi delle coppie connesse sono rappresdentati tramite un vettore ‘id’ di N elementi.
L’operazione di ‘find’ è un semplice riferimento alla cella del vettore id[indice] e ha costo unitario.
L’operazione di Union consiste nella scansione del vettore per cambiare gli elementi che valgono ‘p’ in ‘q’ con costo lineare nella dimensione del vettore.

Nella quick find è l’operazione di find ad essere lineare nella dimensione del vettore in quanto dato un elemento, per comprendere a quale insieme appartiene, occorre percorrere una ‘catena’ di oggetti, mentre l’operazione di union è una semplice assegnazione con valore ‘q’ all’ultima cella della catena percorsa nell’operazione di find.

→ Equazione alle ricorrenze del MergeSort

T(n) = 2T(n / 2) + n dove ‘a’ che è il fattore di divisione indica il numero di sottoproblemi da analizzare ad ogni ricorsione, mentre ‘b’ = 2 indica la dimensione dei due sottoproblemi generati.
La dimensione in questo caso ‘ dimezzata.
La combinazione C(n) = n in quanto occorre ricombinare tutti gli elementi del vettore.
Risolvendo l’equazione alle ricorrenze otteniamo una complessità ottima per quanto riguarda gli algoritmi basati sul confronto O(nlogn).

→ grammatica di una stringa con operandi e operatori

Data una stringa contenente operatori e operandi, si dice ‘arity’ il numero di operandi appartenenti a ciascun operatore.
Limitando la arity a due possiamo quindi rappresentare le espressioni aritmetiche tramite alberi binari, dove le foglie rappresentano gli operandi mentre la radice l’operatore che li correla.

→ struttura per rappresentarla (albero)

L’albero è quindi binario.

→ algoritmi di visita (prefisso, infisso, postfisso)

E poiché è un albero è possibile visitarlo tramite i tre algoritmi di attraversamento che sono pre-order, post-order e in-order.
Visitando l’albero in pre-order otteniamo un’espressione aritmetica in notazione prefissa, che è scarsamente utilizzata. Visitandolo in post-order otteniamo un’espressione aritmetica in notazione post-fissa o polacca inversa (reverse polish notation) che permette di risolvere l’espressione avendo un solo stack. Visitando l’albero in-order otteniamo nvece l’espressione aritmetica così come la costruiamo secondo le regole matematiche di precedenza.

→ Double hashing, perchè è migliore del linear chaning

Nel doube hashing la complessità per la ricerca di un elemento è minore rispetto al linear chaining.
Nel double hashing per un ‘hit’ è infatti 1/alpha * ln (1/1-alpha) mentre per un miss è i/1-alpha.
Nel linear chaining invece, nel caso in cui tutti gli elemti siano andati in collisione e quindi siano stati inseriti tutti nella stessa lista allora la complessità è lineare.

→ componenti fortemente connesse

Una componente connessa invece è un sottografo non orientato, quindi un sottoinsieme di archi non orientati e un sottoinsieme di vertici che presenta la proprietà di mutua raggiungibilità per ogni coppia di vertici che la compongono.
Per trovare le componenti fortemente connesse si esegue l'algoritmo di Kosaraju su un grafo orientato.
Esso consiste nella trasposizione del grafo e della visita di quest'ultimo in profondità. Data tale visita, seguendo l'ordine decrescente dei vertici per tempo di completamente si esegue una seconda visita in profondità questa volta però del grafo originale, costruendo gli alberi della visita ricorsiva si costruiscono allo stesso tempo le componenti fortemente connesse.

→ Come funziona una tabella ad accesso diretto

Una tabella ad accesso diretto è una particolare tabella di simboli implementata tramite vettore. Essa, per definizione, permette un accesso diretto O(1) in fase di inserimento, in fase di ricerca e in fase di cancellazione. Questo è possibile perché la dimensione della tabella corrisponde alla dimensione dell'insieme universo, permettendo una corrispondenza biunivoca tra un oggetto appartenente all'insieme universo e un intero tra 0 e la dimensione dello stesso universo – 1.

→ perchè non è sempre possibile utilizzarla

Il problema di queste tabelle è nella dimensione di memoria occupata, che corrisponde alla dimensione dell'insieme universo. In pratica queste tabelle sono utilizzabili solo se l'insieme universo è relativamente piccolo poiché si ha uno spreco di memoria se il numero di chiavi effettivamente inserito nella tabella è molto minore della dimensione della stessa.
Abbiamo un trade-off tra efficienza nel tempo e spreco di memoria.

→ tutti i tipi di hashing

Abbiamo vari tipi di Hashing, nati per poter risolvere al meglio il problema delle collisioni.
. Il linear chaining permette di allocare in una singola cella della tabella di hash più chiavi utizzando una lista per ogni cella.
Se non vi è collisione con altre chiavi allora la chiave viene inserita direttamente nella sua posizione, mentre se è presente collisione, viene inserita in testa alla lista della cella in cui era previsto l'inserimento.
Questo è inoltre l'unico approccio all'hash che permette di avere una dimensione della tabella di simboli minore o uguagale al numero di chiavi che abbiamo previsto di inserire.
In genere, per mantenere una buona complessità, si sceglie come dimensione della tabella il più piccolo numero primo M maggiore o uguale al numeor massimo delle chiavi fratto 5 o 10 (in modo che, in media, le liste siano lunghe 5 o 10).

. L'open addressing con linear probing permette, invece, l'inserimento di una sola chiave in ciascun elemento della tabella. In caso di collisione, viene incrementato l'indice relativo alla chiave da inserire fino a che non viene trovata una posizione vuota in cui inserire tale elemento.
Con l'open addressing prendiamo come dimensione della tabella il più piccolo numero primo M maggiore uguale al doppio del massimo numero delle chiavi presenti.

→ horner
Il metodo di valutazione Horner permette, in una funzioe di hash, di valutare piccole stringhe secondo un polinomio.
Viene infatti splittata la stringa in tutti i suoi caratteri, moltiplicando ciascun carattere per la base della codifica ASCII 128 (in realtà 127 poiché è un numero primo) oppurtanemetne elevata alla potenza corrispondente al grado del polinomio. Questo metodo porta però a vari inconvenienti,

come all’onerosa operazione di elevamento a potenza e l’elevato valore numerico dell’intero finale che rappresenterà il termine di questa valutazione, per stringhe troppo grandi si rischia di avere molto velocemente un overflow, rendendo inutilizzabile il metodo.

→ c'è qualcosa di migliore per le stringhe

Per ovviare al problema si può fare il modulo del risultato con la dimensione della tabella ad ogni passaggio, eliminando eventuali multipli che porterebbero all’overflow.

→ Ciclo e cammino di Hamilton

Un cammino di Hamilton è un cammino che collega un vertice di partenza s a un vertice di destinazione d e che attraversa tutti i vertici che compongono il grafo una e una sola volta.
Si ha un ciclo di Hamilton quando il vertice sorgente corrisponde anche al vertice destinazione.

→ cammino di Eulero

Un cammino di eulero è un cammino che connette due vertici, destinazione e sorgente e che attraversa tutti gli archi disponibili nel grafo una e una sola volta.

→ cos'è il grado di un vertice di un grafo e di un albero

In un albero il grado corrisponde al numero massimo di figli che si diramano da un nodo dell’albero.
In un vertice di un grafo invece il grado è la somma dell’out_degree e dell’in_degree, ovvero la somma tra il numero di archi che escono dal vertice e il numero di archi che convergono nel vertice. Se il grafo è non orientato allora il grado indica semplicemente il numero di archi incidenti su quel determinato vertice.

→ Paradigma di greedy

Il paradigma Greedy permette, in alcuni casi, di arrivare ad una soluzione ottima senza necessariamente esplorare tutto lo spazio delle soluzioni come invece viene fatto nel paradigma divide et impera. In questo approccio, si sceglie la scelta che localmente risulta essere minima in funzioni di costo o massima in funzione di appetibilità, usando allo stesso tempo un approccio incrementale. Si ottiene quindi in questi casi una soluzione in tempi minori, algoritmi Greedy ottimi sono per esempio l’algoritmo di Dijkstra dei cammini minimi o l’algoritmo di scheduling.

→ il paradigma di greedy in kruskal
→ come funziona kruskal

In kruskal, un algoritmo per la ricerca di un albero ricoprente minimo, gli archi del grafo vengono dapprima ordinati in ordine crescente di peso e poi selezionati uno ad uno. Una volta selezionati, tramite un algoritmo union find si capisce se i nodi collegati dall’arco appartengono ad alberi diveri, se si eseguo la union, se no scarto l’arco e passo al successivo.

→ union-find e relativa complessità

Gli algoritmi di Union-Find sono algoritmi utili a risolvere il problema della online connectivity, ovvero per capire, data in input una coppia di nodi, se tale coppia risulta già connessa da un qualsiasi cammino già inserito precedentemente oppure se fornisce un ulteriore informazione inserendo un arco che li colleghi.

Gli algoritmi sono principalemtne 3, dove ognuno di essi predilige in complessità o l'operazione di union o l'operazione di find.
La quick find per esempio opera su un vettore id con gli identificativi dei vertici e predilige l'operazione di find a scapito di una union più complessa.
La find in questo caso si limita a controllare l'insieme di appartenenza dei deu dati in ingresso p e q. Se tali insiemi sono diversi allora la union scandisce l'intero vettore cambiando i riferimenti p in q.
La quick union predilige la union a scapito di una find più complessa.
La find viene fatta percorrendo una catena di riferimenti, è quindi al massimo lineare nel numeor di vertici (sovente meno). La union si occupa invece dell'operazione di costo unitario di modifica del riferimento di fine catena in q.
Esiste poi la weighted quick union, che permette di avere complessità al più logaritmiche ottimizzando l'operazione di quick union e riducendo la dimensione della catena. In questo caso la union fonde l'albero più piccolo in quello più grande.

→ teorema archi sicuri

Dato un grafo non orientato pesato e connesso, chiamiando A il sottoinsieme degli archi, se:
. A è contenuto in un qualche albero ricoprente minimo di G
. S e V – S è un taglio qualuncuw che rispetta A
. l'arco u,v è un arco leggero che attraversa il taglio (S, V – S)

allora l'arco u,v è un arco sicuro per A.

→ componenti connesse

Una componente connessa è un sottografo non orientato, quindi un sottoinsieme di archi non orientati e un sottoinsieme di vertici del grafo di partenzache presenta la proprietà di mutua raggiungibilità per ogni coppia di vertici che la compongono.

→ partition in un BST

La partition di un BST è una particolare operazione sui BST nella quale si fa galleggiare, ovvero portare al posto della radice, una foglia. L'operazione di partition opera usando rotazioni a destra e a sinistra al fine di far galleggiare il nodo. Può essere usata, se il nodo da far galleggiare è la chiave mediana, per ribilanciare il BST.

→ grado di un nodo, albero

In un albero il grado corrisponde al massimo numero di nodi che si radicano da uno specifico nodo. Tale informazione vale per tutto l'albero.

→ rotazione a sinistra in un BST

```
link rotL (link h) {
```

```
link x = h→r;
h→r= x→ l;
x→ l = h;
return h;
}
```

→ ricerca del pivot nel quick sort

Si prende il pivot all'estrema destra del vettore.

→ Caso peggiore per il Quicksort con risoluzione dell'equazione alle ricorrenze associata; componenti fortemente connesse

Il caso peggiore nel Quick sort si ha quando il vettore in input all'algoritmo è già ordinato. Si ottiene in questo caso una complessitò quadratica. Il caso è facilmente evitabile inserendo un flag che indica se il vettore è già ordinato.

$T(n) = T(n - 1) + n$

→ Differenze tra grafo e albero

Un albero ha |V| - 1 archi, ovvero ha un numero di archi tale da poter garantire la non ciclicità, se infatti venisse inserito un arco si avrebbe un ciclo, cosa che porterebbe l'albero ad essere un grafo.

→ quanti cammini ci sono tra i nodi e la radice in un albero

I cammini sono tanti quanti sono i nodi foglia, in un albero binario le foglie sono 2 ^ h, dove h è l'altezza dell'albero

→ tecniche di rappresentazione degli alberi

Posso rappresentare un albero tramite il metodo left child right sibling.

→ Fattore di carico, cos'è, che significato ha, come si riduce nel linear chianing,

Il fattore di carico alpha viene definito come il rapporto tra il numero delle chiavi inserite e la dimensione massima della tabella.
Nel caso del linear chaining per diminuire il fattore di carico o si diminuisce il numero di chiavi da memorizzare o si aumenta la dimensione della tabella.

→ complessità di ricerca nel linear chaining e complessità di inserzione.

Ricerca: caso peggiore O(n);
caso medio O(1+ alpha)
Inserione: O(1)

→ Hashing Universale

→ Cosa è una cosa a priorità, cosa è un bridge, come implemento la coda a priorità, complessità spaziale dell'heap.

→ Dimostrazione del limite inferiore degli algoritmi di ordinamento basati sul confronto. DAG.

→ Calcolo dei Nodi di un albero binario in maniera ricorsiva (voleva sapere il codice), complessità del QuickSort nel caso peggiore e in quello medio, complessità dei BST.[

→ Heap definizione e proprietà altezza minima e masima, BST proprietà, albero completo quasi completo, bilanciato quasi bilanciato e che rapporto c'è tra bilanciato e completo.

→ MST definizione, differenze rispetto a cammini minimi, Djistra.

→ Cancellazione elemento in una tabella di Hash (in tutti i tipi di tabella)

→ Definizione di cammino e cammino semplice; dire perchè in un grafo, dato un percorso v w u, se so che v u è minimo anche v w è un percorso minimo

→ funzione di appetibilità, esempio di problema con appetibilità variabile (huffman). Merge sort, mi ha chiesto se è un bottom-up o top-down. Scrivere merge sort bottom-up.

→ definizione grafo completo, numero minimo di archi che deve avere per essere tale, definizione di multigrafo, definizione di cammino di Eulero

→ Heap, analisi di complessità Omega grande