Esercitazioni di Algoritmi e Strutture Dati

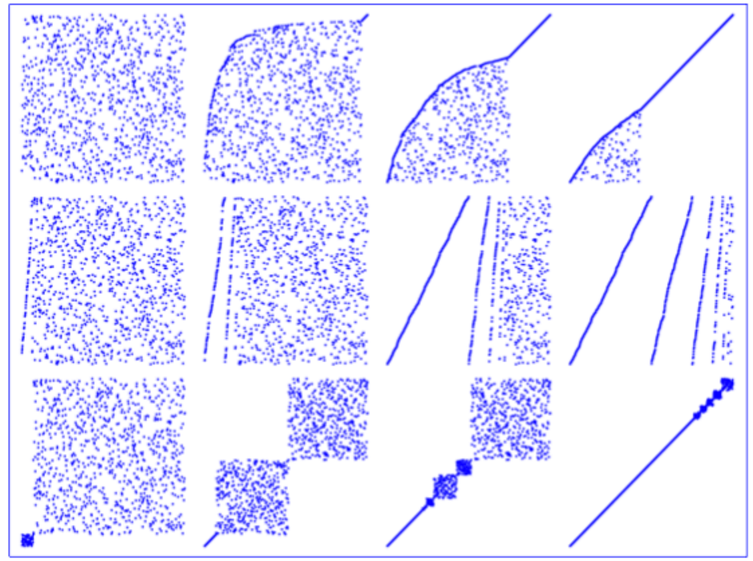
Esercizi di Algoritmi e Strutture Dati

Gli esercizi presentati a lezione sono presi da

https://moreno.marzolla.name/teaching/ASD/

Le slides con gli esercizi sono scaricabili da

https://saveriogiallorenzo.com/teaching/#asd



Moreno Marzolla
http://www.moreno.marzolla.name/

Esercitazione(**Esercizio** *e*) {
  leggere il testo di *e* insieme;
  timeout(10' , { Provare a svolgere *e* singolarmente });
  for((**Studente**, **Soluzione**) (*st*, *so*) : studenti . soluzioni){
    st presenta brevemente la *so* che ha trovato;
  }
  contestualizzazione di *e*, soluzione da 

 e discussione;
}

4. Hashing

Esercizio 4.1

Ricordiamo che una funzione hash $h: U \rightarrow \{0, 1, \ldots, m\text{-}1\}$ è detta *perfetta* se per ogni coppia di chiavi distinte x e y, si ha $h(x) \neq h(y)$.

1. Scrivere un algoritmo per il calcolo di una funzione hash perfetta in cui l'insieme universo sia costituito da sequenze di cinque caratteri scelti tra i seguenti: 'A', 'C', 'T', 'G' e il valore di m sia minimo possibile.
2. Determinare il costo computazionale dell'algoritmo di cui al punto precedente.

8. Grafi

Esercizio 8.1.1

Si consideri un grafo non orientato $G=(V, E)$ in cui a ciascun nodo $v \in V$ è associato un peso reale $w(v)$ (che puo' essere positivo o negativo). Un cammino $<v_1, v_2, \ldots v_k>$ si dice *monotono* se $w(v_1) < w(v_2) < \ldots < w(v_k)$. In altre parole in un cammino monotono i pesi dei nodi attraversati devono essere in ordine strettamente crescente.

1. Dimostrare che se $<v_1, v_2, \ldots v_k>$ è un cammino monotono, allora è aciclico

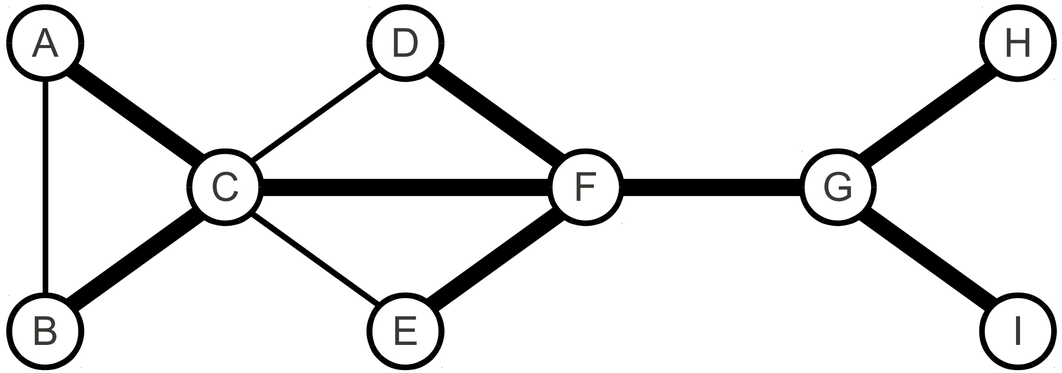
Esercizio 8.1.2-3

Si consideri un grafo non orientato $G=(V, E)$ in cui a ciascun nodo $v \in V$ è associato un peso reale $w(v)$ (che puo' essere positivo o negativo). Un cammino $<v_1, v_2, \ldots v_k>$ si dice *monotono* se $w(v_1) < w(v_2) < \ldots < w(v_k)$. In altre parole in un cammino monotono i pesi dei nodi attraversati devono essere in ordine strettamente crescente.

2. Descrivere un algoritmo efficiente che, dato in input un grafo non orientato $G=(V, E)$ con nodi pesati, e due nodi $s, d \in V$, restituisce *true* se e solo se esiste un cammino monotono che inizia dalla sorgente s e termina nella destinazione d. L'algoritmo deve anche stampare i nodi che compongono tale cammino (i nodi possono essere stampati nell'ordine $v_1, v_2, \ldots v_k$ oppure nell'ordine inverso $v_k, v_{k-1}, \ldots v_1$)
3. Determinare il costo computazionale dell'algoritmo descritto al punto 2, motivando la risposta

Esercizio 8.3

Considerare il seguente grafo non orientato:



1. Gli archi in grassetto possono rappresentare un albero di visita ottenuto mediante una visita **in profondità** del grafo? In caso affermativo specificare il nodo di inizio della visita, e rappresentare il grafo mediante liste di adiacenza in modo tale che l'ordine in cui compaiono gli elementi nelle liste consenta all'algoritmo DFS di produrre esattamente l'albero mostrato.

2. Gli archi in grassetto possono rappresentare un albero di visita ottenuto mediante una visita **in ampiezza** del grafo? In caso affermativo specificare il nodo di inizio della visita, e rappresentare il grafo mediante liste di adiacenza in modo tale che l'ordine in cui compaiono gli elementi nelle liste consenta all'algoritmo DFS di produrre esattamente l'albero mostrato.

Esercizio 8.4

Si consideri un grafo orientato $G=(V, E)$ con n nodi, numerati da 1 a n. Il grafo è rappresentato mediante una matrice di adiacenza.

1. Scrivere un algoritmo per calcolare l'indice di un nodo avente grado uscente massimo (ossia l'indice di un nodo avente massimo numero di archi uscenti).
2. Analizzare il costo computazionale dell'algoritmo proposto.

5. Divide-et-impera

Esercizio 5.1.1

Si consideri il seguente problema: dato un array $A[1..n]$ di $n \geq 1$ valori reali non ordinati, restituire il valore minimo in A.

1. Scrivere un algoritmo *ricorsivo* di tipo *divide et impera* per risolvere il problema di cui sopra; non è consentito usare variabili globali.

Esercizio 5.1.2-3

Si consideri il seguente problema: dato un array $A[1..n]$ di $n \geq 1$ valori reali non ordinati, restituire il valore minimo in A.

2. Calcolare la complessità asintotica dell'algoritmo proposto, motivando la risposta;
3. Dare un limite inferiore alla complessità del problema, nell'ipotesi in cui l'algoritmo risolutivo si basi solo su confronti. Giustificare la risposta.

Esercizio 5.5.1

Una inversione in un array $V[1..n]$ di n numeri reali è una coppia di indici i, j tali che $i < j$ e $V[i] > V[j]$.

1. Scrivere un algoritmo iterativo che dato un array V calcola il numero di inversioni presenti in V in tempo $O(n^2)$.

Esercizio 5.5.2

Una inversione in un array $V[1..n]$ di n numeri reali è una coppia di indici i, j tali che $i < j$ e $V[i] > V[j]$.

2. Scrivere un algoritmo divide-et-impera che dato un array V calcola il numero di inversioni in tempo $O(n \log n)$. *Suggerimento: considerare un algoritmo simile a* MERGESORT*; in particolare, la fase in cui si combinano i risultati parziali può essere implementata in modo simile alla operazione merge di* MERGESORT.

6. Greedy

Esercizio 6.1.1

Disponiamo di un tubo metallico di lunghezza L. Da questo tubo vogliamo ottenere al più n segmenti più corti, aventi rispettivamente lunghezza $S[1]$, $S[2]$, …, $S[n]$. Il tubo viene segato sempre a partire da una delle estremità, quindi ogni taglio riduce la sua lunghezza della misura asportata.

1. Scrivere un algoritmo efficiente per determinare il numero massimo di segmenti che è possibile ottenere. Formalmente, tra tutti i sottoinsiemi degli n segmenti la cui lunghezza complessiva sia minore o uguale a L, vogliamo determinarne uno con cardinalità massimale.

Esercizio 6.1.2

Disponiamo di un tubo metallico di lunghezza L. Da questo tubo vogliamo ottenere al più n segmenti più corti, aventi rispettivamente lunghezza $S[1]$, $S[2]$, …, $S[n]$. Il tubo viene segato sempre a partire da una delle estremità, quindi ogni taglio riduce la sua lunghezza della misura asportata.

2. Determinare il costo computazionale dell'algoritmo di cui al punto 1.

Esercizio 6.3

Supponiamo di avere $n \geq 1$ oggetti, ciascuno etichettato con un numero da 1 a n;. l'oggetto i-esimo ha peso $p[i] > 0$. Questi oggetti vanno inseriti all'interno di scatoloni identici, disponibili in numero illimitato, ciascuno in grado di contenere un numero arbitrario di oggetti purché il loro peso complessivo sia minore o uguale a C. Si può assumere che tutti gli oggetti abbiano peso minore o uguale a C. I pesi sono valori reali arbitrari. Vogliamo definire un algoritmo che disponga gli oggetti negli scatoloni in modo da cercare di minimizzare il numero di scatoloni utilizzati. Questo genere di problema è noto col nome di *bin packing problem* ed è computazionalmente intrattabile nel caso generale; di conseguenza, ci accontentiamo di un algoritmo semplice che produca una soluzione non necessariamente ottima.

1. Scrivere un algoritmo basato sul paradigma greedy che, dato il vettore dei pesi $p[1..n]$ e il valore C, restituisce il numero di scatoloni che vengono utilizzati.
2. Calcolare il costo computazionale dell'algoritmo proposto.

Esercizio 6.3