Esercitazioni di Algoritmi e Strutture Dati

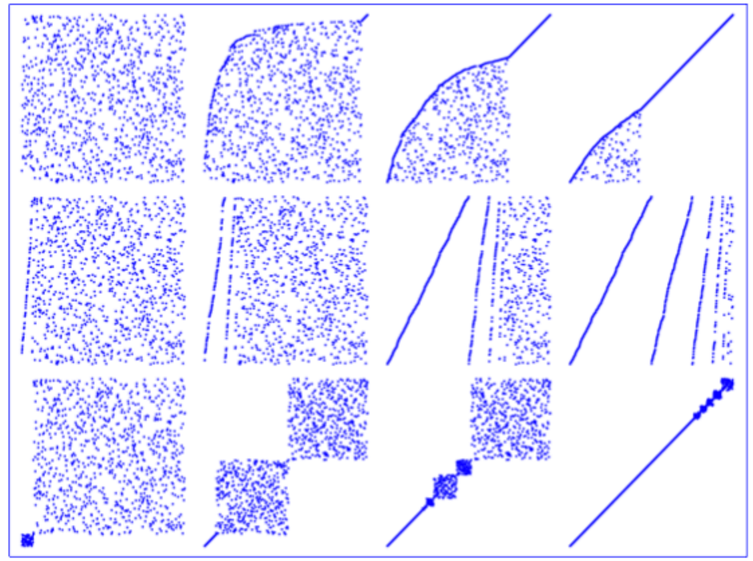
Esercizi di Algoritmi e Strutture Dati

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;
}

7. Dynamic Programming

Esercizio 7.1.1

Un distributore di bibite contiene al suo interno n monete i cui valori sono rispettivamente $c[1]$, $c[2]$, …, $c[n]$. Tutti i valori sono interi positivi; è possibile che più monete presenti nel distributore abbiano lo stesso valore. Si consideri il problema di decidere se è possibile erogare, in qualsiasi modo, un resto *esattamente uguale a* R utilizzando un opportuno sottoinsieme delle n monete a disposizione; R è un intero positivo.

1. Descrivere un algoritmo efficiente per decidere se il problema ammette una soluzione oppure no.

Esercizio 7.1.2

Un distributore di bibite contiene al suo interno *n* monete i cui valori sono rispettivamente *c*[1], *c*[2], …, *c*[*n*]. Tutti i valori sono interi positivi; è possibile che più monete presenti nel distributore abbiano lo stesso valore. Si consideri il problema di decidere se è possibile erogare, in qualsiasi modo, un resto *esattamente uguale a R* utilizzando un opportuno sottoinsieme delle *n* monete a disposizione; *R* è un intero positivo.

2. Determinare il costo computazionale dell'algoritmo descritto al punto 1, motivando la risposta
3. Modificare l'algoritmo di cui al punto 1 per determinare anche quali sono le monete da erogare per produrre il resto *R*. Si noti che non è necessario erogare il resto con il numero minimo di monete: è sufficiente erogarlo in modo qualsiasi.

Esercizio 7.1.3

Un distributore di bibite contiene al suo interno n monete i cui valori sono rispettivamente $c[1]$, $c[2]$, …, $c[n]$. Tutti i valori sono interi positivi; è possibile che più monete presenti nel distributore abbiano lo stesso valore. Si consideri il problema di decidere se è possibile erogare, in qualsiasi modo, un resto *esattamente uguale a* R utilizzando un opportuno sottoinsieme delle n monete a disposizione; R è un intero positivo.

3. Modificare l'algoritmo di cui al punto 1 per determinare anche quali sono le monete da erogare per produrre il resto R. Si noti che non è necessario erogare il resto con il numero minimo di monete: è sufficiente erogarlo in modo qualsiasi.

Esercizio 7.2.1

Supponiamo di avere n file aventi rispettivamente dimensione $F[1]$, $F[2]$, …, $F[n]$; le dimensioni sono numeri interi strettamente positivi e sono espresse in MB. Disponiamo di un CD-ROM avente capacità 650MB; sfortunatamente, il CD-ROM potrebbe non essere sufficientemente capiente per memorizzare tutti gli n file.

1. Scrivere un algoritmo efficiente per determinare il *numero massimo di file* che è possibile memorizzare sul CD-ROM senza eccederne la capacità. Non è richiesto che l'algoritmo stampi anche quali file memorizzare.

Esercizio 7.2.2

Supponiamo di avere n file aventi rispettivamente dimensione $F[1]$, $F[2]$, …, $F[n]$; le dimensioni sono numeri interi strettamente positivi e sono espresse in MB. Disponiamo di un CD-ROM avente capacità 650MB; sfortunatamente, il CD-ROM potrebbe non essere sufficientemente capiente per memorizzare tutti gli n file.

2. Analizzare il costo computazionale dell'algoritmo proposto.

10. Grafi — MST & SPT

Esercizio 10.A

Definire una versione “lazy” dell’algoritmo di Prim per la computazione del minimum spanning tree di un grafo che salta l’aggiunta degli archi ineleggibili dalla lista di priorità anziché della versione “eager” che li rimuove all’atto dell’aggiunta di nuovi archi.

Analizzare il costo computazionale dell’algoritmo proposto.

Esercizio 10.B

Definire una versione “lazy” dell’algoritmo di Dijkstra per la computazione del cammino minimo (shortest path) in un grafo, adattando l’algoritmo lazy di Prim per la computazione del minimum spanning tree di un grafo.

Analizzare il costo computazionale dell’algoritmo proposto.