

*Diario delle lezioni di Algoritmi e Strutture Dati (modulo I), a.a. 2019/20.*

1. (6/10/20). Introduzione al corso. Motivazioni e concetti fondamentali. Un primo esempio: il problema di trovare una moneta falsa (più pesante) fra  $n$  monete usando una bilancia a due piatti.
2. (8/10/20). Il problema del calcolo dell' $n$ -esimo numero di Fibonacci. Un algoritmo numerico e un algoritmo ricorsivo. Analisi della complessità temporale dell'algoritmo ricorsivo. Un algoritmo iterativo di complessità temporale  $O(n)$  e di complessità spaziale  $O(n)$  (Fibonacci3). Portare la memoria a  $O(1)$ : Fibonacci4. Introduzione informale alla notazione asintotica. Algoritmo con complessità  $O(\log n)$  per il calcolo dell' $n$ -esimo numero di Fibonacci. Discussione della complessità spaziale degli algoritmi ricorsivi Fibonacci2 e Fibonacci6.
3. (13/10/20) Modello di calcolo RAM. Costi uniformi e logaritmici. Complessità caso peggiore, migliore, medio. Notazioni asintotiche:  $O$ -grande,  $\Omega$ -grande,  $\Theta$ .  $O$ -piccolo,  $\Omega$ -piccolo. Definizioni e semplici esempi. Proprietà. Usare la notazione asintotica nelle analisi della complessità computazionale degli algoritmi.
4. (15/10/20) Analisi della complessità nel caso medio: un esempio. Il problema della ricerca di un elemento in un insieme: ricerca sequenziale e ricerca binaria. Equazioni di ricorrenza. Metodo dell'iterazione. Metodo che usa l'albero della ricorsione.
5. (20/10/20) Ancora sulle equazioni di ricorrenza. Metodo della sostituzione. Teorema Fondamentale delle Ricorrenze (Master). Semplici esempi. Quando non si può applicare. Metodo del cambiamento di variabile.
6. (24/10/19). Il Problema dell'ordinamento. Un algoritmo semplice ma inefficiente: il Selection Sort. Un algoritmo migliore: il MergeSort. Un altro algoritmo che usa la tecnica divide et impera: il QuickSort: analisi del caso peggiore, migliore, e intuizioni sul caso medio. Discussione versione randomizzata del QuickSort e differenza fra complessità nel caso medio e tempo atteso di un algoritmo randomizzato.
7. (27/10/20). Progettare algoritmi efficienti attraverso la progettazione di strutture dati efficienti. Un esempio: l'HeapSort - che ordina in loco  $n$  elementi in tempo  $O(n \log n)$  nel caso peggiore.
8. (29/10/20). Delimitazioni superiori e inferiori di algoritmi e problemi. Un lower bound alla complessità temporale necessaria per ordinare  $n$  elementi (per una classe di algoritmi ragionevoli, quelli basati su confronti). Un algoritmo veloce per ordinare interi "piccoli": IntegerSort.
9. (03/11/20). Ancora algoritmi di ordinamento non basati su confronti. Una variante dell'IntegerSort per ordinare  $n$  record con chiavi intere: BucketSort. Un

algoritmo veloce per ordinare interi “grandi”: il RadixSort. Discussione del seguente esercizio: dato un array di  $n$  interi compresi fra 1 e  $k$ , costruire in tempo  $O(n+k)$  un *oracolo* (struttura dati) che sia in grado di rispondere in tempo costante a domande del tipo "quanti interi nell'array sono compresi fra  $a$  e  $b$ ?"(Esercizio e soluzioni a fine delle slide sull'IntegerSort).

10. (5/11/20). Esercitazione. Esercizio: dimostrare o confutare una relazione asintotica (Es. 1). Esercizio di progettazione di un algoritmo che, dato un vettore ordinato  $A$  di  $n$  interi distinti e un valore  $x$ , trova (se esistono) due elementi di  $A$  che sommano a  $x$ . Soluzione banale con complessità quadratica, soluzione di complessità  $O(n \log n)$  e soluzione con tempo  $O(n)$  (Es. 2).
11. (10/11/20). Esercitazione. Primo esercizio: dato un array di  $n$  numeri *unimodale*, progettare un algoritmo con complessità  $o(n)$  che trova il massimo e uno con complessità  $o(n \log n)$  che lo ordina (Es. 3). Secondo esercizio: dato un vettore  $A$  di  $n$  numeri, progettare un algoritmo che in tempo  $O(n)$  trova due indici  $i$  e  $j$  con  $i < j$  che massimizzano  $A[j]-A[i]$  (Es. 4).
12. (12/11/20). Strutture dati elementari: rappresentazioni indicizzate e rappresentazioni collegate. Implementazione di un dizionario con array ordinato/non ordinato e lista ordinata/non ordinata. Rappresentazioni di alberi. Algoritmi di visita di un albero: profondità versione iterativa, profondità versione ricorsiva (preordine, postordine, ordine simmetrico), ampiezza. Algoritmo per calcolare l'altezza di un albero.
13. (17/11/20). Esercitazione sulle visite di alberi. Progettazione di un algoritmo che, preso un albero con valori e colori (rosso e nero), trova il valore del cammino rosso di tipo nodo-radice di valore massimo (Es 5). Altro esercizio: progettare un algoritmo che, preso un albero e in intero  $h$ , restituisce il numero di nodi dell'albero di profondità almeno  $h$  (Es 6). Altro esercizio: preso un albero binario con valori, calcola il numero di nodi per cui la somma dei valori degli antenati è uguale alla somma dei valori dei discendenti (Es. 7).
14. (19/11/20). Il problema del Dizionario. Alberi binari di ricerca. Definizione. Visita in ordine simmetrico di un BST. Ricerca, inserimento, cancellazione (ricerca del massimo, del minimo, del predecessore e del successore di un nodo).
15. (24/11/20). Il problema del Dizionario: secondo episodio. Alberi AVL: definizione ed esempi. Dimostrazione della delimitazione superiore dell'altezza di un albero AVL (che usa la nozione di albero di Fibonacci). Operazioni sugli alberi AVL: search, insert, delete.
16. (26/11/20). Esercitazione. Progettare un algoritmo che, dato un vettore ordinato  $A[1:n]$  di  $n$  bit, trova il numero  $k$  di zero presenti in  $A$ . Algoritmo con complessità  $O(\log n)$ . Un miglior algoritmo con tempo  $O(\log k)$  (Es. 8). Progettare un algoritmo con complessità lineare che, dato un vettore  $A[1:n]$  di  $n$  bit, trova l'indice  $k$  tale che il numero di zeri in  $A[1:k]$  è uguale al numero di uni in  $A[k+1:n]$  (Es. 9).

17. (01/12/20). Il problema della Coda con priorità. d-Heap, Heap Binomiali, (cenni sugli) Heap di Fibonacci e complessità ammortizzata.
18. (03/12/20). I Grafi (diretti, non diretti, pesati). Nozioni preliminari. Cammini, distanze, diametro. Alberi. Grafi Euleriani. I grafi come linguaggio potente per descrivere scenari e problemi. Esempi di scenari/problematiche descrivibili come grafi/problemi su grafi (reti stradali/di trasporto, reti sociali, reti “delle dipendenze”).
19. (10/12/20). Strutture dati per rappresentare un grafo. Matrice di adiacenza e Liste di adiacenza. Visite di un grafo. Visita in ampiezza (BFS): cammini minimi da una sorgente. Visita in profondità (DFS): uscire da un labirinto.
20. (15/12/20). Usi meno scontati della visita DFS. Catalogare per tipo gli archi del grafo. Individuare un ciclo in grafi diretti. Grafi diretti aciclici (DAG) e ordinamento topologico. Usare la visita DFS per trovare un ordinamento topologico di un DAG. Componenti fortemente connesse: un algoritmo lineare per calcolarle.
21. (17/12/20). Cammini minimi in grafi pesati. Il problema del calcolo dei cammini minimi a singola sorgente. Un algoritmo veloce quando il grafo ha pesi non negativi: l'algoritmo di Dijkstra.
22. (12/01/21). Correzione Problem Set 1.
23. (14/01/21). Correzione Problem Set 2.
24. (19/01/21). Simulazione prova d'esame scritta e discussione risultati e modalità d'esame.