Esercizi di teoria

Analisi della complessità

Equazione alle ricorrenze

1. Esegui 3 iterazioni dell'equazione alle ricorrenze.
    a. Divide and conquer:

$$\begin{cases} T(n) = T(\frac{n}{x}) + c(n) & n > 1, x > 2 \\ T(1) = 1 & n = 1 \end{cases}$$

    b. Decrease and conquer:

$$\begin{cases} T(n) = T(n - x) + c(n) & n > 1, x > 2 \\ T(1) = 1 & n = 1 \end{cases}$$

2. Sostituisci la terza equazione ottenuta nella seconda, successivamente sostituire la seconda ottenuta nella prima.
3. Trovare una legge che può essere definita attraverso una sommatoria, e definire l'indice alla quale terminerà.
    a. Divide and conquer:

$$\frac{n}{x^i} = 1 \Rightarrow i = \log_x n$$

    b. Decrease and conquer:

$$n - ix = 1 \Rightarrow i = \frac{n-1}{x}$$

4. Risolvere la sommatoria ottenuta.
5. Il valore più alto ottenuto (non negativo) rispetto ad n sarà quello utilizzato per la definizione della complessità.

Algoritmi di ordinamento ricorsivi

QuickSort

1. Definisci il pivot come l'elemento all'ultimo indice del vettore.
2. Definisci l'indice i come quello del primo elemento del vettore restante, e l'indice j come quello dell'ultimo elemento del vettore restante.
    a. Scorri l'indice i a destra fino a trovare un elemento minore del pivot.
    b. Scorri l'indice j a sinistra fino a trovare un elemento minore del pivot.
3. Confrontiamo gli indici.
    a. Se $i \geq j$, passa al punto successivo.
    b. Se $i < j$ scambia gli elementi all'indice i e all'indice j e ripeti dal punto 2.
4. Scambia l'elemento all'indice j con quello all'indice del pivot.
5. Utilizzando l'elemento pivot come divisore, dividi in due sotto-vettori il vettore principale, ed esegui il QuickSort su entrambi.
6. Termina quando il vettore è ordinato.

MergeSort

1. Definisci l'indice m a metà del vettore, dove l indica l'indice limite sinistro e r indica l'indice del limite destro.

$$m = \lfloor (l + r)/2 \rfloor$$

2. Esegui il merge sort sui sotto-vettori limitati dagli indici (l, m) e dagli indici $(m + 1, r)$.
3. Ripeti fino ad avere vettori di 1 elemento, che sono, per definizione, ordinati.
4. Unisci i sotto-vettori ordinati sinistro e destro, ed esegui l'ordinamento.
5. Continua fino a risalire l'intero albero di divisione ed ottenere l'intero vettore ordinato.

Programmazione dinamica

Parentesizzazione

1. Definisci le matrici $A_1, A_2, ..., A_n$, di dimensioni $p_0 \times p_1, p_1 \times p_2, ..., p_{n-1} \times p_n$, con $p_0, p_1, ..., p_n$ dimensioni delle matrici.
2. Disegna la matrice superiore indicizzata da 1 a n per la valutazione della parentesizzazione ottima, con le prime n celle con valore 0.
3. Disegna la matrice superiore indicizzata da 2 a n per la stampa della soluzione, con le prime $n-1$ celle con valori da 1 a $n-1$.
4. Definiamo i valori della parentesizzazione per intervalli di dimensione 2, con $k = 1 + x + y$, con $x = \{0, 1, ..., n-1\}$ e $y = \{0, 1, ..., n-1\}$ e prendere i valori minori tra le scelte, inserendole all'indice $(2 + y, 1 + y)$ nella prima matrice.

   $$m_{k(k+1)} = m_{kk} + m_{(k+1)(k+1)} + p_{k-1}p_kp_{k+1}$$

   Nella seconda matrice, inserire l'indice k alla posizione $(2 + y, 1 + y)$.
5. Definiamo i valori della parentesizzazione per intervalli di dimensione 3, con $k = 1 + x + y$, con $x = \{0, 1, ..., n-2\}$ e $y = \{0, 1, ..., n-2\}$ e prendere i valori minori tra le scelte, inserendole all'indice $(3 + y, 1 + y)$.

   $$m_{k(k+2)} = \begin{cases} m_{kk} + m_{(k+1)(k+2)} + p_{k-1}p_kp_{k+2} \\ m_{k(k+1)} + m_{(k+2)(k+2)} + p_{k-1}p_{k+1}p_{k+2} \end{cases}$$

   Nella seconda matrice, inserire l'indice k alla posizione $(3 + y, 1 + y)$.
6. Ripetere fino a riempire le due matrici.
7. Per stampare la soluzione, a partire dall'elemento $(n, 1)$, definendo il primo indice come i e il secondo indice come j, calcolare il valore $m = \lfloor (i + j)/2 \rfloor$.
   a. Se $i \neq j$, ricorrere sugli indici (m, i) e $(j, m + 1)$.
   b. Se $i = j$, stampare la matrice con indice i.
   c. Ripetere fino a terminazione dell'albero di ricorsione.

Longest Increasing Sequence

1. Definire una tabella con 3 righe: la prima contiene una sequenza, la seconda contiene la lunghezza L della sotto-sequenza e la terza contiene l'indice del valore precedente P nella sotto-sequenza.
    a. I valori della lunghezza sono inizializzati a -1.
    b. I valori del precedente sono inizializzati a -1.
2. Il primo valore nella sequenza viene inizializzato con valori $L = 1, P = -1$.
3. Dal secondo valore in poi, si controllano tutti i valori precedenti nella sequenza.
    a. Se il valore è minore dell'elemento con cui si sta confrontando, allora si passa al successivo.
    b. Se la lunghezza attuale è minore della lunghezza di uno qualsiasi degli elementi, allora si cambia il valore di L con la lunghezza dell'altro elemento incrementato di 1 e si cambia il valore di P con l'indice dell'elemento con cui si è confrontata la lunghezza.
4. Ripetere il punto 3 fino al termine della sequenza.
5. Per stampare la soluzione, a partire dall'elemento con valore L massimo, si stampa la sequenza relativa, definita dai predecessori di quell'elemento.

Paradigma greedy

Codice di Huffman

1. Ordinare gli elementi da inserire nel codice di Huffman rispetto ai loro valori.
2. Unire i primi due simboli correnti in un albero.
    a. La radice è definita dalla somma delle frequenze dei due simboli.
    b. L'elemento sinistro diventa l'elemento sinistro della radice.
    c. L'elemento destro diventa l'elemento sinistro della radice.
    d. L'albero risultante viene inserito in maniera ordinata nella sequenza ordinata rispetto al suo valore.
3. Ripetere il passaggio 2 con tutti gli elementi, siano essi simboli o alberi.
4. Termina quando è presente solo un elemento nella sequenza.

Massimo numero di attività mutuamente compatibili

1. Ordinare le attività in ordine di tempi di fine crescenti.
2. Prendere la prima attività, e confrontarla con tutte le altre.
    a. Se si intersecano, allora la seconda attività non si può prendere;
    b. Se non si intersecano, allora la seconda attività si può prendere.
3. La soluzione è definita mentre si percorrono le attività.

Tabella di Hash

Ricordarsi la definizione del fattore di carico:

$$\alpha = \frac{N}{M}$$

Inserimento in tabella di Hash

Linear Chaining

1. Definiamo la funzione di Hashing.

$$h(k) = k\%M$$

2. Si applica la chiave di Hashing al valore k definito dalla funzione di Hashing.
    a. Se la cella con indice $h(k)$ è vuota, allora inserisci il valore in lista.
    b. Se non è vuota, allora si inserisce il valore in testa alla lista.

Linear Probing

1. Definiamo la funzione di Hashing.

$$h(k) = (k + i)\%M$$

2. Si applica la chiave di Hashing al valore k definito dalla funzione di Hashing, con inizialmente $i = 0$.
    a. Se la cella con indice $h(k)$ è vuota, allora inserisci il valore nella cella.
    b. Se non è vuota, allora tenta l'inserimento con $i = i + 1$, e si ripete fino al successo.

Quadratic Probing

1. Definiamo la funzione di Hashing, scegliendo c_1 e c_2 dai possibili valori $\{0, 0.5, 1\}$.

$$h(k) = (k + c_1 i + c_2 i^2)\%M$$

2. Si applica la chiave di Hashing al valore k definito dalla funzione di Hashing, con inizialmente $i = 0$.
    a. Se la cella con indice $h(k)$ è vuota, allora inserisci il valore nella cella.
    b. Se non è vuota, allora tenta l'inserimento con $i = i + 1$, e si ripete fino al successo.

Double Hashing

1. Definiamo la funzione di Hashing, scegliendo $h_1(k)$ e $h_2(k)$ e A numero primo rispetto a M.

$$h_1(k) = K\%M \qquad\qquad h_2(k) = 1 + K\%A$$

$$h(k) = [h_1(k) + i \cdot h_2(k)]\%M$$

2. Si applica la chiave di Hashing al valore k definito dalla funzione di Hashing, con inizialmente $i = 0$.
    a. Se la cella con indice $h(k)$ è vuota, allora inserisci il valore nella cella.
    b. Se non è vuota, allora tenta l'inserimento con $h_1(k) = h(k)$ e $i = i + 1$, e si ripete fino al successo.

Heap

Inserimento in coda a priorità

1. L'inserimento in un heap di dati avviene inserendo l'elemento dopo quello dell'ultimo indice.
2. Si confronta la chiave dell'elemento inserito con il suo padre.
    a. Se è maggiore, allora vengono scambiati.
    b. Se è minore, allora è già nella posizione corretta.

Cancellazione in coda a priorità

1. Si scambia l'elemento alla radice dell'albero con l'ultimo elemento presente nell'heap.
2. Si applica la trasformazione in heap, mantenendo l'elemento alla fine dell'heap appena scambiato nella stessa posizione.

Cambio di priorità

1. Si cambia la priorità dell'elemento interessato.
2. Si porta l'elemento nella posizione corretta, facendolo salire o scendere in base al valore della chiave.

Trasformazione in heap

1. Le foglie sono già heap. Si parte dai padri delle foglie.
2. Si confronta la radice con il figlio sinistro.
    a. Se l'elemento è minore del figlio sinistro, allora si propone il figlio sinistro come nuova radice del sotto-albero.
    b. Altrimenti si propone l'elemento come radice del sotto-albero.
3. Si confronta l'elemento candidato con il figlio destro.
    a. Se l'elemento candidato è minore del figlio destro, allora si propone il figlio destro come nuova radice del sotto-albero.
    b. Altrimenti si propone l'elemento come radice del sotto-albero.

4. L'elemento candidato diventa la radice del sotto-albero, sostituendo la radice attuale.
5. Si ripete con tutti i nodi dal basso verso l'alto.

Heapsort

1. Si scambia l'elemento alla radice dell'albero con l'ultimo elemento presente nell'heap.
2. Si applica la trasformazione in heap, mantenendo l'elemento alla fine dell'heap appena scambiato nella stessa posizione.
3. Applica questo procedimento per ogni elemento fino ad ottenere l'ordinamento.

Binary Search Tree

Visita pre-order, in-order, post-order

Tutte le visite partono dalla radice dell'albero.

1. La visita pre-order è definita attraverso la visita della radice, seguita dalla visita del figlio sinistro, seguita dalla visita del figlio destro.
2. La visita in-order è definita attraverso la visita del figlio sinistro, seguita dalla visita della radice, seguita dalla visita del figlio destro.
3. La visita post-order è definita attraverso la visita del figlio sinistro, seguita dalla visita del figlio destro, seguita dalla visita della radice.

Espressione in forma prefissa, infissa e postfissa

Tutti i nodi che non sono foglie sono operatori. Le foglie sono i valori utilizzati in combinazione con gli operatori.

1. La forma prefissa è definita attraverso una visita pre-order di un albero delle espressioni.
2. La forma infissa è definita attraverso una visita in-order di un albero delle espressioni.
3. La forma postfissa è definita attraverso una visita post-order di un albero delle espressioni.

Inserimento in foglia

1. Se l’albero è vuoto, si inserisce l’elemento da inserire nella radice.
2. Se non è vuoto, si inserisce nell’albero iniziando il confronto con la radice.
    a. Se l’elemento è maggiore della radice, viene portato l’inserimento nell’elemento di destra.
    b. Se l’elemento è minore della radice, viene portato l’inserimento nell’elemento di sinistra.
3. Si ripete fino a quando non viene trovata un nodo sentinella.

Inserimento in radice

1. Se l’albero è vuoto, si inserisce l’elemento da inserire nella radice.
2. Se non è vuoto, si inserisce nell’albero inserendolo in foglia.
3. Si confronta la foglia appena inserita con quella padre, allo scopo di mantenere le proprietà dei BST.
    a. Se l’elemento non è in posizione ed è il figlio destro, allora si ruota verso sinistra.
        i. Il padre del figlio destro diventa il padre del padre.
        ii. Il figlio sinistro del figlio destro diventa il figlio destro del padre.
    b. Se l’elemento non è in posizione ed è il figlio sinistro, allora si ruota verso destra.
        i. Il padre del figlio sinistro diventa il padre del padre.
        ii. Il figlio destro del figlio sinistro diventa il figlio sinistro del padre.
    c. Se l’elemento è in posizione non si fa nulla.

Ricerca della chiave k

1. La ricerca parte dalla radice del BST.
   a. Se la chiave ricercata è maggiore della chiave della radice, allora si ricerca nel figlio sinistro.
   b. Se la chiave ricercata è minore della chiave della radice, allora si ricerca nel figlio destro.
   c. Se la chiave ricercata è uguale alla chiave della radice, è stato trovato l'elemento.
2. Si ripete fino a quando si arriva a un nodo sentinella oppure si trova la chiave ricercata.

Partizionamento intorno alla k-esima chiave

1. Si ricerca la chiave di rango k.
2. Si effettuano rotazioni coerenti fino a portare l'elemento alla radice.

Cancellazione

1. Si ricerca la posizione della chiave e si rimuove l'elemento.
2. Se l'elemento eliminato è una radice, si effettua un partizionamento della radice dell'albero destro.
3. L'elemento sinistro della radice diventa l'albero sinistro ottenuto dalla rimozione dell'elemento.

Selezione di una k-esima chiave

Per la selezione è necessario avere un BST esteso, con la cardinalità presente in ogni nodo e i nodi sentinella alle foglie.

1. A partire dalla radice si effettua il confronto tra il rango e la cardinalità del figlio sinistro.
   a. Se il rango è maggiore della cardinalità, allora si procede nel figlio sinistro con rango k.
   b. Se il rango è minore della cardinalità, allora si procede nel figlio destro con rango $k = k - c - 1$.
2. Si ripete fino ad avere k = 0.

Interval Binary Search Tree

Inserimento in foglia

L'inserimento in foglia avviene allo stesso modo dell'inserimento in foglia nel BST, utilizzando come chiave il tempo d'inizio dell'intervallo.

Grafo

Visita del grafo

Visita in profondità

1. Si parte da un determinato nodo x, assegnando il tempo di inizio 0 e incrementandolo.
2. Attraverso un determinato ordine, al prossimo nodo che si attraversa viene assegnato il tempo di inizio, incrementandolo.
    a. Se un nodo non è ancora stato visitato quando viene raggiunto, allora l'arco attraverso il quale è stato raggiunto è un arco T.
    b. Se un nodo è stato visitato quando viene raggiunto e il tempo di terminazione è minore del tempo di terminazione del nodo precedente, allora l'arco attraverso il quale è stato raggiunto è un arco B.
    c. Se un nodo è stato visitato quando viene raggiunto e il tempo di scoperta è minore del tempo di scoperta del nodo precedente, allora l'arco attraverso il quale è stato raggiunto è un arco F.
    d. Tutti gli altri archi sono C.
3. Si continua fino a quando sono presenti nodi non ancora visitati che sono raggiungibili dal nodo attuale. In questo caso, si imposta il tempo di terminazione al valore del tempo attuale, successivamente incrementandolo.
4. Si ripete fino alla terminazione di ogni nodo.

Visita in ampiezza

1. Si parte da un determinato nodo x, che sarà la radice dell'albero di visita in ampiezza.
2. I figli della radice sono i nodi raggiungibili da x che non sono ancora stati visitati.
3. Si ripete per ogni nodo.

Ordinamento topologico del DAG

1. Si effettua una visita in profondità.
2. Si ordinano i nodi per ordine inverso di tempo di terminazione di ogni nodo.

Punti di articolazione

1. Si applica la visita in profondità al grafo, ottenendo l'albero di visita in profondità con i relativi archi T, B, F, C.
2. I punti di articolazione sono tutti i nodi che mantengono connesso il grafo, anche definiti come nodi nella quale i nodi figli non hanno archi di tipo B che connettono ad un antenato.

Cammini massimi del DAG

1. Si effettua l'ordinamento topologico del DAG dal nodo x.
2. Inizialmente si impostano i cammini massimi dal nodo x a tutti gli altri nodi come $-\infty$.
3. Si calcolano i cammini massimi dal nodo x ad ogni altro nodo, a partire dal primo nodo dell'ordinamento topologico.

Algoritmo di Kosaraju

1. Si effettua una visita in profondità su un grafo trasposto.
    a. Il grafo trasposto ha gli archi inversi rispetto al grafo originale.
2. Si effettua la stessa visita a partire dall'ordine inverso dei tempi di terminazione di ogni nodo.
    a. Ogni volta che viene effettuata una visita in profondità, si inizia una componente fortemente connessa.
3. Si ripete fino ad avere tutte le componenti fortemente connesse.

Minimum Spanning Tree

Algoritmo di Prim

1. A partire da un determinato nodo, si effettuano i tagli sui nodi che raggiungono nodi adiacenti non ancora visitati.
2. Si sceglie, per ogni taglio effettuato, l'arco a peso minore.
3. Si ripete fino ad avere un MST.

Algoritmo di Kruskal

1. Si inizia con l'insieme degli archi a pesi minori.
2. Si prendono tutti gli archi che non comprendono la creazione di un ciclo nel grafo.
3. Si ripete con il prossimo insieme degli archi a pesi minori fino ad avere un MST.

Cammini minimi

Algoritmo di Djikstra

1. Si inizia definendo una soluzione vuota, con una coda a priorità che contiene ogni nodo e il costo del cammino da un nodo x scelto verso ogni nodo, impostandolo inizialmente come ∞ e il costo del cammino dal nodo x al nodo x come 0.
2. Si estrae l'elemento a costo minore dalla coda a priorità e si rilassano gli archi, riordinando la coda a priorità per costi crescenti.
3. Si ripete fino ad avere la coda a priorità vuota, ed avendo quindi tutti i cammini minimi da x verso ogni altro nodo.

Algoritmo di Bellman-Ford

1. Si crea una tabella di dimensione $(N+1) \times N$, con N numero di vertici del grafo. Si utilizzeranno le colonne per definire i passi, mentre le righe si usano per la distanza minima dal nodo scelto fino al nodo indicato nella riga.
2. Si ordinano gli archi in ordine lessicografico e si inizializza la prima colonna della tabella con valore ∞ e 0 sul nodo scelto.

3. Si guarda ogni arco nell'ordine definito. Se è presente un cammino con distanza minore per un determinato nodo, allora si inserisce nella tabella, al determinato nodo, una nuova distanza minima.
4. Si ripete fino a non avere variazioni della distanza minima. Se si hanno variazioni anche al passo $N + 1$, allora il grafo contiene un ciclo a peso negativo, e quindi il cammino minimo perde di senso.