Algoritmi e Strutture Dati

AA 2018/2019

Paolo Baldan
Dipartimento di Matematica
Università di Padova

27 settembre 2021

Segue una raccolta di esercizi svolti, suddivisi approssimativamente per aree tematiche. Gli esercizi sono spesso accompagnati da una bozza di soluzione, che contiene una indicazione su come procedere nella soluzione dell'esercizio. Spesso la soluzione è solo abbozzata ed è quindi da intendersi come una guida alla soluzione. In particolare, quando la domanda richieda definizioni o illustrazioni di procedimenti senza contributi originali, la soluzione non le include. Inoltre il documento è di recente redazione e può contenere sviste e piccoli errori. Segnalazioni in questo senso sono benvenute.

Si ringraziano per le segnalazioni (in ordine temporale): Bogdan Stanciu, Bianca Andreea Ciuche, Alberto Schiabel, Alberto Miola, Elisabetta Piombin, Jordan Gottardo, Sara Righetto, Ilaria Peron.

1 Limiti asintotici e ricorrenze

Domanda 1 Risolvere la ricorrenza $T(n) = 4\,T(n/2) + n \log n$ utilizzando il master theorem.

Domanda 2 Mostrare che la ricorrenza $T(n) = T(n/2) + T(n/4) + n$ ammette soluzione $T(n) = \Theta(n)$ utilizzando il metodo di sostituzione.

Domanda 3 Risolvere la ricorrenza $T(n) = 4T(n/2) + n^2\sqrt{n}$ utilizzando il master theorem.

Domanda 4 Risolvere la ricorrenza $T(n) = T(n-2) + 2n$ utilizzando il metodo di sostituzione per dimostrare un limite asintotico stretto.

Domanda 5 Risolvere la ricorrenza

$$T(n) = \begin{cases} 3 & \text{se } n = 0 \\ T(n-1) + 2 & \text{se } n > 0 \end{cases}$$

utilizzando il metodo di sostituzione per determinare una soluzione esatta (non asintotica).

Domanda 6 Sia data la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 3T(n/5) + T(n/6) + n & \text{se } n > 1 \end{cases}$$

Si fornisca un limite asintotico stretto per la soluzione.

Domanda 7 Sia data la seguente equazione di ricorrenza:

$$T(n) = 5T(\lfloor n/3 \rfloor) + 2n^2$$

Si fornisca un limite asintotico stretto per la soluzione.

Domanda 8 Sia data la seguente equazione di ricorrenza:

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

Si dimostri che $T(n) = O(n \log n)$.

Domanda 9 Sia data la seguente equazione di ricorrenza:

$$T(n) = T(n/2) + T(\sqrt{n}/2) + 2n$$

Si dimostri che $T(n) = \Theta(n)$.

Domanda 10 Si consideri la ricorrenza

$$T(n) = T\left(\frac{4n}{5}\right) + \frac{n}{2} + \log n$$

Fornire limite asintotico stretto per la soluzione.

Domanda 11 Si dimostri che la ricorrenza che segue ha soluzione $T(n) = \Theta(n)$

$$T(n) = \frac{1}{2}T(n-1) + n$$

Domanda 12 Dare la definizione di $\Omega(f(n))$. Dimostrare che la ricorrenza che segue ha soluzione $T(n) = \Omega(2^n)$

$$T(n) = \sum_{k=1}^{n-1} T(k)\,T(n-k)$$

Domanda 13 Dare la definizione di $O(f(n))$. Dimostrare che la ricorrenza che segue ha soluzione $T(n) = O(n)$

$$T(n) = \frac{2}{3}T(n-1) + 2n$$

Domanda 14 Dare una soluzione asintotica per la ricorrenza $T(n) = 3T(n/2) + n(n+1)$.

Domanda 15 Data la ricorrenza $T(n) = T(n/2) + T(n/3) + \sqrt{n} + 2$ dimostrare che ha soluzione $T(n) = O(n)$. Il limite è stretto, ovvero vale anche $T(n) = \Omega(n)$? [Questa seconda parte della domanda, per un errore nella scelta dei coefficienti risultava più complessa di quanto preventivato, quindi non è stata considerata nella valutazione. Allego comunque la soluzione per completezza.]

Domanda 16 Data la ricorrenza $T(n) = 5T(n/3) + (n-2)^2$, trovare la soluzione asintotica.

Domanda 17 Dare la definizione di $\Omega(f(n))$. Mostrare che se $f(n) = \Omega(g(n))$ e $g(n) = \Omega(h(n))$ allora $f(n) = \Omega(h(n))$.

Domanda 18 Dare la definizione di $\Theta(f(n))$. Mostrare che $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$.

2 Divide et impera e Ricorsione

Esercizio 1 Dato un array di interi $A[1..n]$, chiamiamo *gap* un indice $i \in [1, n)$ tale che $A[i+1] - A[i] > 1$.

i. Mostrare per induzione su n che un array $A[1..n]$ tale che $A[n] - A[1] \geq n$ (quindi $n \geq 2$) contiene almeno un gap.

ii. Fornire lo pseudocodice di una procedura ricorsiva divide et impera *gap* che dato un array $A[1..n]$ tale che $A[n] - A[1] \geq n$ restituisce un gap in A.

iii. Valutare la complessità della funzione, utilizzando il master theorem.

Esercizio 2 Scrivere una procedura di tipo divide et impera *over* che dato un array di interi distinti $A[1..n]$, ordinato in modo crescente, e un intero x restituisce l'indice del più piccolo elemento in A strettamente maggiore di x. Se nessun elemento di A soddisfa la condizione, si restituisca $n + 1$. Valutare la complessità dell'algoritmo.

Esercizio 3 Realizzare una procedure di tipo divide et impera `Max(A,p,r)` per trovare il massimo nell'array `A[p,r]`. Si assuma che l'array non sia vuoto (`p` $\leq$ `r`). Scrivere lo pseudocodice e valutare la complessità con il master theorem.

Esercizio 4 Sia `A[1..n]` un array di interi distinti ordinato in senso crescente. Dimostrare che dato un qualunque indice `i`, se `A[i] > i` allora `A[j]>j` per ogni `j>i` e analogamente se `A[i] < i` allora `A[j]<j` per ogni `j<i`.

Utilizzare l'osservazione per realizzare una funzione `Fix(A)` che dato l'array di interi `A[1..n]` ordinato senza ripetizioni restituisce un indice `i` tale che `A[i]=i`, se esiste, e `0` altrimenti. Valutarne la complessità.

Domanda 19 Scrivere una funzione ricorsiva `subseq(X,Y,m,n)` che date due sequenze $X[1..m]$ e $Y[1..n]$, di lunghezza m e n rispettivamente, verifica se X è una sottosequenza di Y e restituisce un valore booleano conseguente. Valutarne la complessità.

Esercizio 5 Un array $A[1..n]$ di numeri si dice *alternante* se non ha elementi contigui identici (ovvero per ogni $i \leq n-1$ vale $A[i] \neq A[i+1]$) e inoltre per ogni $i \leq n-2$, vale che $a_i < a_{i+1} > a_{i+2}$ oppure $a_i > a_{i+1} < a_{i+2}$. Ad esempio gli array [1, 2, -1, 3, 2] e [5, 1, 2, -1, 3, 2] sono alternanti, mentre non lo sono [1, 2, 3] e [1, 1, 2]. Scrivere una funzione ricorsiva `alt(A,n)` che dato un array $A[1..n]$ di numeri verifica se è alternante. Valutarne la complessità.

3 Ordinamento

Esercizio 6 Fornire lo pseudocodice di una procedura `min(A,B)` che dati due array `A` e `B` che siano uno la permutazione dell'altro ne trova l'elemento minimo confrontando esclusivamente elementi di `A` ed elementi di `B` (non si possono confrontare due elementi di `A` o due elementi di `B` tra loro, e non si possono fare copie degli array, ovvero la funzione deve operare con spazio costante). Valutare la complessità della funzione.

Domanda 20 Realizzare una fDunzione `RevCountingSort(A,B,n,k)` che, dato un array `A[1..n]` contenente interi nell'intervallo `[0..k]`, restituisce in `B[1..n]` una sua permutazione ordinata in modo decrescente utilizzando una variante del counting sort. Valutarne la complessità.

Esercizio 7 Realizzare una funzione `Anagramma(x,y)` che date due stringhe `x` e `y` sull'alfabeto { `0, 1` }, verifica se `x` è un anagramma di `y` restituendo conseguentemente `true` o `false`. Una stringa è vista come un array di caratteri, con lunghezza data dall'attributo `len`. Ad esempio, la stringa `x` è la sequenza di caratteri `x[1] x[2]` ... `x[x.len]`. Valutare la complessità.

Domanda 21 Realizzare una funzione `Double(A,n)` che, dato un array `A[1,n]` ordinato in senso crescente, verifica se esiste una coppia di indici `i, j` tali che $\mathtt{A}[\mathtt{j}] = 2 * \mathtt{A}[\mathtt{i}]$. Restituisce la coppia se esiste e (0,0) altrimenti. Scrivere lo pseudocodice e valutare la complessità.

Esercizio 8 Si dica che un array `A[1..n]` è bi-ordinato se esiste un indice `k` tale che `A[1..k]` è ordinato in senso crescente e `A[k..n]` è ordinato in senso decrescente. Realizzare un algoritmo `top(A,n)` che dato in input un array `A` bi-ordinato con elementi distinti ne determina il valore massimo. Ad esempio su `A = (1 2 3 14 13 4)` restituisce `14`. Valutarne la complessità.

Esercizio 9 Scrivere una funzione `perm(m,n)` che dati due numeri interi m e n, maggiori o uguali di 0, verifica se uno dei due numeri può essere ottenuto permutando le cifre dell'altro. Ad esempio 915 e 159 sono uno la permutazione dell'altro mentre 911 e 19 no. Attenzione al ruolo degli zeri, ad es. 150 è la permutazione di 51, dato che $51 = 051$. Valutarne la complessità.
(*Suggerimento*: Il counting sort può fornire una ispirazione.)

4 Heap

Domanda 22 Scrivere una funzione $sndmin(A)$ che dato in input un array A organizzato a min-heap, restituisce il successore della radice, ovvero il minimo elemento dello heap maggiore della radice. Se un tale elemento non esiste genera un errore. Assumere che A sia non vuoto e gli elementi in A siano tutti distinti.

Domanda 23 Scrivere una funzione `IsMaxHeap(A)` che dato in input un array di interi `A[1..n]` che verifica se `A` è organizzato a max-heap e ritorna un corrispondente valore booleano. Valutarne la complessità.

Domanda 24 Fornire lo pseudocodice della procedura `HeapExtractMin(A)` per estrarre il minimo elemento da un min-heap A, realizzato tramite un array come visto a lezione (la dimensione corrente dello heap è data dall'attributo $A.heapsize$). Discuterne la complessità.

Domanda 25 Dare la definizione di max-heap. Dato un array `A[1..12]` con sequenza di elementi `[60, 6, 45, 95, 30, 24, 15, 80, 19, 38, 21, 70]` si indichi il risultato della procedura `BuildMaxHeap` applicata ad `A`. Si descriva sinteticamente come si procede per arrivare al risultato.

5 Alberi e ricorsione

Domanda 26 Dato un albero nel quale i nodi contengono una chiave, si definisca *costo* di un cammino dalla radice ad una foglia, come la somma delle chiavi dei nodi che compaiono nel cammino. Scrivere una funzione `MaxPath(T)` che opera nel modo seguente. Prende in input un albero binario T, con radice $T.root$, e nodi x che hanno come campi $x.k$, $x.l$ e $x.r$, ovvero una chiave, il puntatore al figlio sinistro e destro, rispettivamente. Resituisce la il costo del cammino di costo massimo dalla radice ad una foglia. Valutarne la complessità.

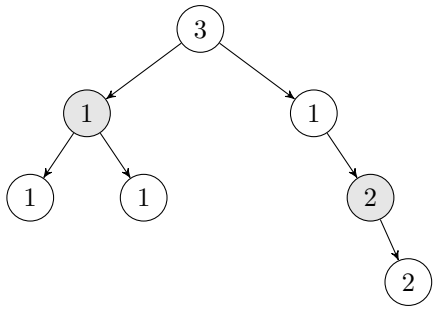
Domanda 27 Realizzare una procedura `Level(T)` che dato un albero binario `T`, con radice `T.root`, e nodi `x` con campi `x.left`, `x.right` e `x.key`, rispettivamente figlio destro, figlio sinistro e chiave intera, ritorna il numero di nodi per i quali la chiave `x.key` è minore o uguale al livello del nodo (la radice ha livello 0, i suoi figli livello 1 e così via). Valutare la complessità.

Domanda 28 Sia T un albero binario i cui nodi x hanno i campi *x.left*, *x.right*, *x.key*. L'albero si dice un *sum-heap* se per ogni nodo x, la chiave di x è maggiore o uguale sia alla somma delle chiavi nel sottoalbero sinistro che alla somma delle chiavi nel sottoalbero destro.

Scrivere una funzione `IsSumHeap(T)` che dato in input un albero T verifica se `T` è un sum-heap e ritorna un corrispondente valore booleano. Valutarne la complessità.

Esercizio 10 Un nodo x di un albero binario T si dice *fair* se la somma delle chiavi nel cammino che conduce dalla radice dell'albero al nodo x (escluso) coincide con la somma delle chiavi nel sottoalbero di radice x (con x incluso). Realizzare un algoritmo ricorsivo *printFair(T)* che dato un albero T stampa tutti i suoi nodi fair. Supporre che ogni nodo abbia i campi *x.left*, *x.right*, *x.p*, *x.key*. Valutare la complessità dell'algoritmo.

Un esempio: i nodi grigi sono fair



Esercizio 11 Sia dato un albero i cui nodi contengono una chiave intera $x.key$, oltre ai campi $x.l$, $x.r$ e $x.p$ che rappresentano rispettivamente il figlio sinistro, il figlio destro e il padre. Si definisce *grado di squilibrio* di un nodo il valore assoluto della differenza tra la somma delle chiavi nei nodi foglia del sottoalbero sinistro e la somma delle chiavi dei nodi foglia del sottoalbero destro. Il grado di squilibrio di un albero è il massimo grado di squilibrio dei suoi nodi.

Fornire lo pseudocodice di una funzione `sdegree(T)` che calcola il grado di squilibrio dell'albero T (si possono utilizzare funzioni ricorsive di supporto). Valutare la complessità della funzione.

Esercizio 12 Si consideri un albero binario T, i cui nodi x hanno i campi $x.l$, $x.r$, $x.p$ che rappresentano il figlio sinistro, il figlio destro e il padre, rispettivamente. Un *cammino* è una sequenza di nodi $x_0, x_1, \ldots, x_n$ tale che per ogni $i = 1, \ldots, n$ vale $x_{i+1}.p = x_i$. Il cammino è detto *terminabile* se $x_n.l = nil$ oppure $x_n.r = nil$. Diciamo che l'albero è 1-bilanciato se tutti i cammini terminabili dalla radice hanno lunghezze che differiscono al più di 1. Scrivere una funzione `bal1`(T) che dato in input l'albero T verifica se è 1-bilanciato e ritorna un corrispondente valore booleano. Valutarne la complessità.

Esercizio 13 Sia T un albero binario i cui nodi x hanno i campi *x.left*, *x.right*, *x.key*. L'albero si dice *k-bounded*, per un certo valore k, se per ogni nodo x la somma delle chiavi lungo ciascun cammino da x ad una foglia è minore o uguale a k.

Scrivere una funzione `k-bound(T,k)` che dato in input un albero T e un valore k verifica se `T` è k-bounded e ritorna un corrispondente valore booleano. Valutarne la complessità.

Domanda 29 Scrivere una funzione *complete*(T) che dato in input un albero binario verifica se è completo (ovvero ogni nodo interno ha due figli e tutte le foglie hanno la stessa distanza dalla radice).

Domanda 30 Scrivere una funzione `diff(T)` che dato in input un albero binario `T` determina la massima differenza di lunghezza tra due cammini che vanno dalla radice ad un sottoalbero vuoto. Ad esempio sull'albero ottenuto inserendo 1, 2 e 3 produce 2, su quello ottenuto inserendo 2, 1, 3 produce 0. Valutarne la complessità.

6 Tabelle Hash

Domanda 31 Si consideri una tabella hash di dimensione $m = 8$, e indirizzamento aperto con doppio hash basato sulle funzioni $h_1(k) = k \ mod \ m$ e $h_2(k) = 1 + k \ mod \ (m - 2)$. Si descriva in dettaglio come avviene l'inserimento della sequenza di chiavi: 12, 3, 22, 14, 38.

Domanda 32 Si consideri una tabella hash di dimensione $m = 8$, e indirizzamento aperto con doppio hash basato sulle funzioni $h_1(k) = k \ mod \ m$ e $h_2(k) = 1 + 2 * (k \ mod \ (m - 3))$. Si descriva in dettaglio come avviene l'inserimento della sequenza di chiavi: 34, 12, 18, 9, 42.

Domanda 33 Si consideri una tabella hash di dimensione $m = 8$, e indirizzamento aperto con doppio hash basato sulle funzioni $h_1(k) = k \ mod \ m$ e $h_2(k) = 1 + k \ mod \ (m - 2)$. Si descriva in dettaglio come avviene l'inserimento della sequenza di chiavi: 14, 22, 10, 16, 8.

Domanda 34 Si consideri una tabella hash di dimensione $m = 8$, gestita mediante chaining (liste di trabocco) con funzione di hash $h(k) = k \ mod \ m$. Si descriva in dettaglio come avviene l'inserimento della sequenza di chiavi: 14, 10, 22, 18, 19.

Domanda 35 Si consideri una tabella hash di dimensione $m = 9$, e indirizzamento aperto con doppio hash basato sulle funzioni $h_1(k) = k \ mod \ m$ e $h_2(k) = 1 + k \ mod \ (m - 2)$. Si descriva in dettaglio come avviene l'inserimento della sequenza di chiavi: 12, 3, 22, 14, 38.

7 Alberi Binari di Ricerca e Red Black Trees

Domanda 36 Realizzare una funzione $pred(x)$ che dato in input un nodo x, di un albero binario di ricerca T, restituisce il predecessore di x (oppure nil, se il predecessore non esiste). Come a lezione, supporre che ogni nodo abbia i campi *x.left*, *x.right*, *x.p*, *x.key*.

Domanda 37 Scrivere una funzione `IsABR(A)` che dato in input un array di interi `A[1..n]`, interpretato come albero binario (come nel caso degli heap, ogni `A[i]` è un nodo con figlio sinistro e destro `A[2i]` e `A[2i+1]`) verifica se `A` è un albero binario di ricerca. Valutarne la complessità.

Domanda 38 Si consideri una variante degli alberi binari di ricerca nella quale i nodi `x` hanno un campo `x.pred` (predecessore) invece che il campo `x.p` (parent). Realizzare la procedura di `Insert(T,z)` che inserisce un nodo `z` nell'albero. Valutarne la complessità.

Esercizio 14 Realizzare una procedura `BST(A)` che dato un array `A[1..n]` di interi, ordinato in modo crescente, costruisce un albero binario di ricerca di altezza minima che contiene gli elementi di `A` e ne restituisce la radice. Per allocare un nuovo nodo dell'albero si utilizzi una funzione `mknod(k)` che dato un intero `k` ritorna un nuovo nodo con `x.key=k` e figlio destro e sinistro `x.left = x.right = nil`. Valutarne la complessità.

Esercizio 15 Si consideri una estensione degli alberi binari di ricerca nei quali ogni nodo x ha anche un campo booleano $x.even$ che vale *true* o *false* a seconda che la somma delle chiavi nel sottoalbero radicato in x sia pari o dispari. Realizzare la procedura `Insert(T,z)` di modo che mantenga correttamente aggiornato anche il campo *even*. Valutarne la complessità.

Esercizio 16 Scrivere una funzione `range(T,k1,k2)` che dato un albero binario di ricerca T e due interi k_1 e k_2 tali che $k_1 \leq k_2$, stampa, in ordine crescente, tutte le chiavi k contenute nell'albero tali che $k_1 \leq k \leq k_2$. Valutarne la complessità.

Esercizio 17 Scrivere una funzione `RBTree(T)` che dato in input un albero binario di ricerca T, i cui nodi x, oltre ai campi $x.key$, $x.left$ e $x.right$, hanno un campo $x.col$ che può essere B (per "black") oppure R (per "red"), verifica se questo è un Red-Black tree. In caso negativo, restituisce -1, altrimenti restituisce l'altezza nera della radice. Valutarne la complessità.

8 Programazione Dinamica

Esercizio 18 Dare un algoritmo per individuare, all'interno di una stringa $a_1 \dots a_n$ una sottostringa (di caratteri consecutivi) palindroma di lunghezza massima. Ad esempio, nella stringa "*colonna*" la sottostringa palindroma di lunghezza massima è "*olo*". Più precisamente:

i. dare una caratterizzazione ricorsiva della lunghezza massima $l_{i,j}$ di una sottostringa palindroma di $a_i \dots a_j$;

ii. tradurre tale definizione in un algoritmo (bottom up o top down con memoization) che determina la lunghezza massima;

iii. trasformare l'algoritmo in modo che permetta anche di individuare la stringa, non solo la sua lunghezza;

iv. valutare la complessità dell'algoritmo.

Esercizio 19 Sia data un'espressione $E = x_1\ op_1\ x_2\ op_2\ \dots\ x_{n-1}\ op_{n-1}\ x_n$, con $n \geq 2$, dove ogni x_i è un intero positivo e $op_i \in \{+, *\}$ di somma o di moltiplicazione (dati). Utilizzando la programmazione dinamica si determini una parentetizzazione dell'espressione che rende il valore dell'espressione minimo. Ad esempio, l'input 7+10∗2 può essere parentetizzato come ((7+10)∗2) = 34 oppure come $(7 + (10 * 2)) = 27$. In questo caso, la parentetizzazione desiderata è quindi la seconda. Più precisamente:

i. dare una caratterizzazione ricorsiva del valore minimo $v_{i,j}$ prodotto da una parentetizzazione della sottoespressione $x_i\ op_i \dots op_{j-1}\ x_j$;

ii. tradurre tale definizione in un algoritmo (bottom up o top down con memoization) che determina il valore minimo;

iii. trasformare l'algoritmo in modo che permetta anche di stampare l'espressione;

iv. valutare la complessità dell'algoritmo.

Esercizio 20 Si ricordi che data una sequenza $X = x_1 \dots x_k$, si indica con X_i il prefisso $x_1 \dots x_i$. Una sottosequenza di X è $x_{i_1} \dots x_{i_h}$ con $1 \leq i_1 < i_2 < \dots i_h \leq k$, ovvero è una sequenza ottenuta da X eliminando alcuni elementi. Quando Y è sottosequenza di X si scrive $Y \sqsubseteq X$.

Realizzare un algoritmo che, date due sequenze $X = x_1 \dots x_k$ e $Y = y_1 \dots y_h$ determina una *shortest common supersequence* (SCS) ovvero una sequenza Z, di lunghezza minima, tale che $X \sqsubseteq Z$ e $Y \sqsubseteq Z$. Ad esempio per $X = abf$ e $Y = afgj$ una SCS è $abfgj$.

i. Dare una caratterizzazione ricorsiva della lunghezza $l_{i,j}$ di una SCS di X_i e Y_j e dedurne un algoritmo;

ii. trasformare l'algoritmo in modo che fornisca una SCS di X e Y;

iii. valutare la complessità dell'algoritmo.

Esercizio 21 Si supponga di dover pagare una certa somma s. Per farlo si hanno a disposizione le banconote $b_1, \dots, b_n$ ciascuna di valore $v_1, \dots, v_n$ (numeri naturali). Si vuole determinare, se esiste, un insieme di banconote $b_{i_1}, \dots, b_{i_k}$ che totalizzi esattamente la somma richiesta e che minimizzi il numero k di banconote utilizzate.

i. mostrare che vale la proprietà della sottostruttura ottima e fornire una caratterizzazione ricorsiva del costo $c(s', j)$ della soluzione ottima per il sottoproblema di dare una somma pari a s' con le banconote in $b_1, \ldots, b_j$, con $j \leq n$;

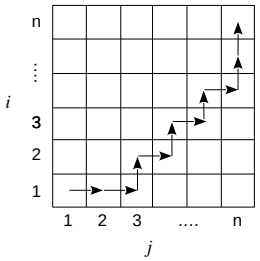
ii. tradurre tale definizione in un algoritmo (bottom up o top down con memoization) che determina il costo della soluzione ottima;

iii. trasformare l'algoritmo in modo che permetta anche di individuare la soluzione, non solo il suo costo;

iv. valutare la complessità dell'algoritmo.

Esercizio 22 Si supponga di avere una scacchiera `n` $\times$ `n`. Si vuole spostare un pezzo dall'angolo in basso a sinistra (1,1) a quello in alto a destra `(n,n)`.

Il pezzo può muoversi di una casella verso l'alto ($\uparrow$) o verso destra ($\rightarrow$). Un passo dalla casella `(i,j)` ha un costo `u(i,j)` se verso l'alto e `r(i,j)` se verso destra. Realizzare un algorimo `MinPath(u,r,n)` che dati in input gli array `u[1..n,1..n]` e `r[1..n,1..n]` dei costi dei singoli passi fornisce il cammino minimo. Più in dettaglio:



i. fornire una caratterizzazione ricorsiva del costo minimo di un cammino `C(i,j)` per andare dalla casella `(i,j)` alla casella `(n,n)`

ii. tradurre tale definizione in un algoritmo `MinPath(u,r,n)` (bottom up o top down con memoization) che determina il costo di un cammino minimo da `(1,1)` a `(n,n)`

iii. trasformare l'algoritmo in modo che stampi la sequenza di passi di costo minimo;

iv. valutare la complessità dell'algoritmo.

Esercizio 23 Sia data una sequenza di città c_1 c_2 ... c_n collegate da un percorso ferroviario. Da ciascuna città c_i è possibile raggiungere c_j, con $i < j$ con un treno diretto. Sapendo che per $i, j \in \{1, \ldots, n\}$, $i < j$ il costo del biglietto del treno diretto da c_i a c_j risulta essere `b[i,j]`, determinare il costo minimo di un tragitto, con possibili cambi intermedi, da c_1 a c_n. Più in dettaglio:

i. fornire una caratterizzazione ricorsiva del costo minimo `min[i]` di un di un tragitto dalla città c_i alla città c_n;

ii. tradurre tale definizione in un algoritmo `MinTrain(b,n)` (bottom up o top down con memoization) che determina il costo di un tragitto di costo minimo dalla città c_1 alla città c_n;

iii. trasformare l'algoritmo di modo che determini anche la sequenza dei cambi necessari per ottenere il tragitto di costo minimo, privilegiando – a parità di costo – soluzioni che minimizzano il numero dei cambi;

iv. valutare la complessità dell'algoritmo.

Esercizio 24 Si supponga che il mondo consista di n stati $S_1 \ldots, S_n$ e che spostarsi dallo stato S_i allo stato S_j imponga l'ottenimento di un visto di ingresso che ha un costo $v_{i,j}$, maggiore di 0. Realizzare un algoritmo che determini una sequenza di stati che conviene percorrere per andare dallo stato S_1 allo stato S_n, minimizzando il costo complessivo dei visti. Più precisamente:

i. Dare una caratterizzazione ricorsiva del costo complessivo minimo v_i dei visti per andare dallo stato S_i allo stato S_n;

ii. Usare la caratterizzazione al punto precedente per ottenere un algoritmo $ESP(v, n)$ che dato l'array dei costi dei visti $v[1..n, 1..n]$ determina il costo minimo dei visti per andare da S_1 a S_n;

iii. trasformare l'algoritmo in modo che determini oltre al costo minimo anche la sequenza degli stati da attraversare per ottenerlo;

iv. valutare la complessità dell'algoritmo.

Esercizio 25 Un marinaio ha n pezzi di corda di lunghezze intere $l_1, \ldots, l_n$ ($l_i \geq 2$) e deve annodarli per ottenere una corda di lunghezza esattamente d. Tenendo conto che fare un nodo richiede una lunghezza pari ad 1 (ad es. se annodo due pezzi lunghi 5 e 7 la corda che ottengo è lunga 11), individuare, se esiste, un insieme minimo di pezzi che annodati producano una corda della lunghezza d desiderata.

Più precisamente:

i. Dare una caratterizzazione ricorsiva del numero minimo $m(i, d')$ di pezzi di corda scelti in $l_1, \ldots, l_i$ che possono essere annodati per produrre la lughezza d' ($+\infty$ se non è possibile ottenere d' con nessuna combinazione);

ii. Usare la caratterizzazione al punto precedente per ottenere un algoritmo $Rope(l, n, d)$ che dato l'array delle lunghezze $l[1..n]$ determina il numero minimo di pezzi da annodare per ottenere d (se esiste);

iii. trasformare l'algoritmo in modo che restituisca oltre al numero anche l'indicazione di quali pezzi usare;

iv. valutare la complessità dell'algoritmo.

Esercizio 26 Siano date n attività $a_1, \ldots, a_n$, che devono essere svolte in un'aula. Ogni attività ha tempo di inizio s_i e tempo di fine f_i, con $s_i < f_i$. Si vuole trovare un sottoinsieme di attività che possano essere svolte nell'aula, quindi senza sovrapposizioni, e che massimizzi il tempo di utilizzo dell'aula.

Più precisamente, assumendo che le attività siano ordinate in modo crescente per tempo di fine:

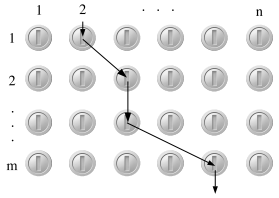
i. dare una caratterizzazione ricorsiva del tempo massimo $u(i, f)$ di utilizzo dell'aula, quando le attività da allocare siano $a_1, \ldots, a_i$ e l'aula sia disponibile fino al tempo f (ovvero nell'intervallo $[0, f)$);

ii. usare la caratterizzazione al punto precedente per ottenere un algoritmo $allocate(s, f, n)$ che dati gli array dei tempi di inizio e fine delle attività $s[1..n]$ e $f[1..n]$ determini il tempo di utilizzo massimo;
Sugg.: è utile osservare che nel calcolo di $u(i, f)$ i valori significativi di f sono i tempi di inizio delle varie attività e quindi è possibile limitarsi a $u(i, j)$, ovvero tempo massimo di utilizzo dell'aula, con attività $a_1, \ldots, a_i$, quando l'aula sia disponibile fino al tempo s_j;

iii. trasformare l'algoritmo in modo che restituisca oltre al tempo anche anche l'indicazione di quali attività allocare;

iv. valutare la complessità dell'algoritmo.

Esercizio 27 Mario deve attraversare una griglia come in figura, dall'alto verso il basso e per farlo, ad ogni passo salta verso il basso di una riga, spostarsi contestualmente a destra di quanto vuole. Un esempio di attraversamento è indicato in figura. Ogni casella contiene una moneta di un certo valore (possibilmente negativo) per cui nell'attraversamento Mario totalizzerà un certo guadagno.



Supponendo che la griglia abbia dimensione $m \times n$ e che per $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$ la casella (i, j) contenga una moneta di valore $V[i, j]$ realizzare un algoritmo che identifica un attraversamento di valore massimo.

Più precisamente:

i. dare una caratterizzazione ricorsiva del guadagno massimo $G[i, j]$ di un attraversamento della sottogriglia $[i..m, j..n]$;

ii. usare la caratterizzazione al punto precedente per ottenere un algoritmo `mario(V,m,n)` che dato l'array `V` con il valore delle monete determini il guadagno massimo di un attraversamento;

iii. trasformare l'algoritmo in modo che restituisca oltre al guadagno anche anche l'indicazione dell'attraversamento da seguire;

iv. valutare la complessità dell'algoritmo.

Esercizio 28 Data una sequenza di numeri $X = x_1 x_2 \ldots x_n$ si vuole determinare una sottosequenza $x_{i_1} x_{i_2} \ldots x_{i_k}$ di X, crescente, ovvero tale che $x_{i_1} \leq x_{i_2} \leq \ldots \leq x_{i_k}$, di lunghezza massima. Ad esempio, se $X = 77\ \ 69\ \ 70\ \ 19\ \ 71\ \ 25\ \ 52\ \ 26\ \ 28\ \ 64$, una sottosequenza crescente di lunghezza massima è $19\ \ 25\ \ 26\ \ 28\ \ 64$

i. fornire una caratterizzazione ricorsiva della lunghezza l_i di una sottosequenza crescente di lunghezza massima che abbia come primo carattere x_i;

ii. tradurre tale definizione in un algoritmo `LIS(X,n)` (bottom up o top down con memoization) che determina la lunghezza di una sottosequenza crescente di lunghezza massima della sequenza $X[1..n]$;

iii. trasformare l'algoritmo in modo individui anche la sottosequenza, non solo la sua lunghezza;

iv. valutare la complessità dell'algoritmo.

Esercizio 29 Date due sequenze $X = x_1 \ldots x_m$ e $Y = y_1 \ldots y_m$ su di un alfabeto $\Sigma = \{a_1, \ldots, a_\ell\}$ si definisce edit distance il costo minimo di un insieme di operazioni che trasforma la sequenza X nella sequenza Y. Le operazioni possibili sono

- *insert*, ovvero l'inserimento di un simbolo in qualche posizione in X (costo 1);

- *delete*, ovvero la cancellazione di un simbolo in qualche posizione in X (costo 1);
- *replace*, ovvero la sostituzione di un simbolo in qualche posizione in X, con costo che dipende dai simboli coinvolti: sostituire il simbolo a con il simbolo b, con $a \neq b$ ha un costo $c(a, b) > 0$.

Ad esempio per trasformare `gesto` in `gelo` posso fare una replace $\mathtt{s} \to \mathtt{l}$ e una delete di `t` con costo $c(\mathtt{s}, \mathtt{l}) + 1$, oppure una delete `s` e una replace $\mathtt{t} \to \mathtt{l}$ con costo $1 + c(\mathtt{t}, \mathtt{l})$, ecc.

Fornire un algoritmo di programmazione dinamica che dati X, Y e la funzione c dei costi di replace (per $a, b \in \Sigma$, $c(a, b)$ è il costo dell'operazione $a \to b$) calcola la edit distance tra X e Y. In dettaglio,

i. indicata con $e(i, j)$ la edit distance tra $X^i = x_1 \dots x_i$ e $Y^j = y_1 \dots y_j$, con $i \in \{0, \dots, m\}$ e $j \in \{0, \dots, n\}$, darne una caratterizzazione ricorsiva;

ii. tradurre tale definizione in un algoritmo (bottom up o top down con memoization) che determina l'edit distance;

iii. trasformare l'algoritmo in modo che fornisca anche le operazioni di edit che trasformano X in Y con costo minimo;

iv. valutare la complessità dell'algoritmo.

Esercizio 30 Nello stato di Frattalia, si è appena insediato il nuovo governo, sostenuto da due partiti, Arancio e Cobalto. La coalizione Arancio-Cobalto ha sottoscritto un contratto di governo che prevede diversi punti, alcuni proposti dagli Arancio altri dai Cobalto. Tuttavia ogni punto ha un costo economico, così che non è possibile realizzarli tutti: il debito pubblico non può oltrepassare una certa soglia. Occorre quindi scegliere un sottoinsieme di punti da realizzare e tale insieme deve essere *bilanciato*, ovvero devono essere scelti in egual numero punti degli Arancio e dei Cobalto.

Si supponga che i punti proposti dal partito Arancio siano `n` con costi $a_1, \dots, a_n$ e analogamente i punti proposti dai Cobalto siano `m` con costi $c_1, \dots, c_m$. Fornire un algoritmo di programmazione dinamica che individui un sottoinsieme bilanciato dei punti del contratto, di dimensione massima e tale che la somma dei costi non oltrepassi un limite fissato d. Più precisamente:

i. dare una caratterizzazione ricorsiva della cardinalità massima $p_{i,j,d}$ di un insieme bilanciato di punti di programma scelti in $a_1, \dots, a_i$, $c_1, \dots, c_j$ con costo $\leq d$;

ii. tradurre tale definizione in un algoritmo (bottom up o top down con memoization) che determina il numero massimo;

iii. trasformare l'algoritmo in modo che fornisca anche i punti del contratto scelti, non solo il loro numero;

iv. valutare la complessità dell'algoritmo.

Esiste la possibilità di applicare un algoritmo greedy?

Esercizio 31 Dare un algoritmo per individuare, all'interno di una stringa $a_1 \dots a_n$ una sottosequenza (quindi una sequenza di caratteri possibilmente non consecutivi) palindroma di lunghezza massima. Ad esempio, nella stringa "*corollario*" la sottosequenza palindroma di lunghezza massima è "*orllro*". Più precisamente:

i. dare una caratterizzazione ricorsiva della lunghezza massima $l_{i,j}$ di una sottosequenza palindroma di $a_i \dots a_j$;

ii. tradurre tale definizione in un algoritmo (bottom up o top down con memoization) che determina la lunghezza massima;

iii. trasformare l'algoritmo in modo che fornisca anche la sottosequenza, non solo la sua lunghezza;

iv. valutare la complessità dell'algoritmo.

Esercizio 32 Realizzare, con tecniche di programmazione dinamica, un algoritmo che dato un array $A[1..n]$, non vuoto, trova un sottoarray non vuoto di somma massima, ovvero due indici i, j con $1 \leq i \leq j \leq n$ tali che $A[i] + A[i+1] + \ldots + A[j]$ sia massima. Ad esempio per [-10, 4, 1, -1, 2, -1] il sottoarray di somma massima è [4, 1, -1, 2]. Più precisamente:

i. indicato con l_j la somma massima di una sottoarray di $A[1..n]$ che termini con $A[j]$ (quindi del tipo $A[i..j]$), darne una caratterizzazione ricorsiva;

ii. tradurre tale definizione in un algoritmo (bottom up o top down con memoization) che determina la somma massima;

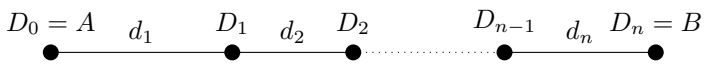
iii. trasformare l'algoritmo in modo che fornisca anche la sottosstringa, non solo la sua somma;

iv. valutare la complessità dell'algoritmo.

Nota: La soluzione proposta deve articolarsi nei passi sopra descritti.

9 Greedy

Esercizio 33 Si supponga di voler viaggiare dalla città A alla città B con un'auto che ha un'autonomia pari a d km. Lungo il percorso si trovano $n-1$ distributori $D_1, \ldots, D_{n-1}$, a distanze di $d_1, \ldots, d_n$ km ($d_i \leq d$) come indicato in figura



L'auto ha inizialmente il serbatoio pieno e l'obiettivo è quello di percorrere il viaggio da A a B, minimizzando il numero di soste ai distributori per il rifornimento.

i. Introdurre la nozione di soluzione per il problema e di costo della una soluzione. Mostrare che vale la proprietà della sottostruttura ottima e individuare una scelta che gode della proprietà della scelta greedy.

ii. Sulla base della scelta greedy individuata al passo precedente, fornire un algoritmo greedy `stop(d,n)` che dato in input l'array delle distanze `d[1..n]` restituisce una soluzione ottima.

iii. Valutare la complessità dell'algoritmo.

Esercizio 34 L'ufficio postale offre un servizio di ritiro pacchi in sede su prenotazione. Il destinatario, avvisato della presenza del pacco, deve comunicare l'orario preciso al quale si recherà allo sportello. Sapendo che gli impiegati dedicano a questa mansione turni di un'ora, con inizio in un momento qualsiasi, si chiede di scrivere un algoritmo che individui l'insieme minimo di turni di un'ora sufficienti a soddisfare tutte le richieste. Più in dettaglio, data una sequenza $\vec{r} = r_1, \ldots, r_n$ di richieste, dove r_i è l'orario della i-ma prenotazione, si vuole determinare una sequenza di turni $\vec{t} = t_1, \ldots, t_k$, con t_j orario di inizio del j-mo turno, che abbia dimensione minima e tale che i turni coprano tutte le richieste.

i. Formalizzare la nozione di soluzione per il problema e il relativo costo. Mostrare che vale la proprietà della sottostruttura ottima e individuare una scelta che gode della proprietà della scelta greedy.

ii. Sulla base della scelta greedy individuata al passo precedente, fornire un algoritmo greedy `time(R, n)` che dato in input l'array delle richieste `r[1..n]` restituisce una soluzione ottima.

iii. Valutare la complessità dell'algoritmo.

Domanda 39 Indicare il codice prefisso ottenuto utilizzando l'algoritmo di Huffmann per l'alfabeto $\{a, b, c, d, e, f, g\}$, supponendo che ogni simbolo appaia con le seguenti frequenze.

a	b	c	d	e	f	g
16	12	2	8	3	9	6

Spiegare il processo di costruzione del codice.

Domanda 40 Indicare il codice prefisso ottenuto utilizzando l'algoritmo di Huffmann per l'alfabeto $\{a, b, c, d, e, f, g\}$, supponendo che ogni simbolo appaia con le seguenti frequenze.

a	b	c	d	e	f	g
5	9	3	3	1	1	6

Spiegare il processo di costruzione del codice.

Domanda 41 Indicare il codice prefisso ottenuto utilizzando l'algoritmo di Huffmann per l'alfabeto $\{a, b, c, d, e, f, g\}$, supponendo che ogni simbolo appaia con le seguenti frequenze.

a	b	c	d	e	f	g
6	21	12	8	3	23	8

Spiegare il processo di costruzione del codice.

Domanda 42 Indicare il codice prefisso ottenuto utilizzando l'algoritmo di Huffmann per l'alfabeto $\{a, b, c, d, e, f, g\}$, supponendo che ogni simbolo appaia con le seguenti frequenze.

a	b	c	d	e	f	g
37	4	12	6	9	17	8

Spiegare il processo di costruzione del codice.

Domanda 43 Indicare il codice prefisso ottenuto utilizzando l'algoritmo di Huffmann per l'alfabeto $\{a, b, c, d, e, f, g\}$, supponendo che ogni simbolo appaia con le seguenti frequenze.

a	b	c	d	e	f	g
10	6	2	8	19	31	15

Spiegare il processo di costruzione del codice.

Domanda 44 Indicare il codice prefisso ottenuto utilizzando l'algoritmo di Huffmann per l'alfabeto $\{a, b, c, d, e, f, g\}$, supponendo che ogni simbolo appaia con le seguenti frequenze.

a	b	c	d	e	f	g
3	8	7	12	6	23	21

Spiegare il processo di costruzione del codice.

Domanda 45 Indicare il codice prefisso ottenuto utilizzando l'algoritmo di Huffmann per l'alfabeto $\{a, b, c, d, e, f, g\}$, supponendo che ogni simbolo appaia con le seguenti frequenze.

a	b	c	d	e	f	g
2	12	16	8	6	9	3

Spiegare il processo di costruzione del codice.

10 Analisi Ammortizzata

Esercizio 35 Progettare una struttura dati per la gestione di un insieme dinamico di interi, con operazioni

- $New(S)$ crea un insieme vuoto;
- $Ins(S, x)$ inserisce l'elemento x nell'insieme S;
- $Half(S)$ cancella da S i $\lceil |S|/2 \rceil$ elementi più piccoli.

Si richiede che una qualsiasi sequenza di n operazioni venga eseguita in tempo $O(n \log n)$.

i. Specificare le strutture dati di supporto utilite e lo pseudo-codice delle operazioni suddette (questo può ridursi ad una chiamata di un'operazione della struttura scelta).

ii. Dimostrare, mediante un'analisi ammortizzata della complessità, che una sequenza di n operazioni costa $O(n \log n)$.

Esercizio 36 Si consideri una struttura che chiamiamo `OderedStack` con le seguenti operazioni:

- `OEmpty(S)`: Ritorna un booleano che dice se la struttura è vuota
- `OPop(S)`: Estrae l'ultimo elemento inserito
- `OTop(S)`: Ritorna il valore dell'ultimo elemento inserito
- `OPush(S,x)`: Inserisce nello stack x, eliminando prima ogni elemento che sia maggiore di x.

Utilizzando una struttura dati stack, fornire una implementazione della suddetta struttura ovvero

i. Specificare come è definita la struttura dati e fornire lo pseudocodice dei metodi elencati.

ii. Mostrare che una sequenza di n operazioni, a partire da struttura vuota, ha costo ammortizzato $O(n)$ (e quindi ogni singola operazione ha costo ammortizzato $O(1)$).

Esercizio 37 Progettare una struttura dati per la gestione di un insieme dinamico di interi, con operazioni

- $New(S)$ crea un insieme vuoto;
- $Ins(S, x)$ inserisce l'elemento x nell'insieme S;
- $Half(S)$ cancella da S i $\lceil |S|/2 \rceil$ elementi più grandi.

Si richiede che una qualsiasi sequenza di n operazioni venga eseguita in tempo $O(n \log n)$.

i. Specificare le strutture dati di supporto utilite e lo pseudo-codice delle operazioni suddette (questo può ridursi ad una chiamata di un'operazione della struttura scelta).

ii. Dimostrare, mediante un'analisi ammortizzata della complessità, che una sequenza di n operazioni costa $O(n \log n)$.

11 B-Alberi

Domanda 46 Dare la definizione di B-albero. Qual è la minima altezza di un B-albero con grado minimo t contenente n chiavi? Motivare le risposte.