Corso di laurea in Informatica
Introduzione agli Algoritmi
A.A. 2024/2025

Esercizi riepilogativi

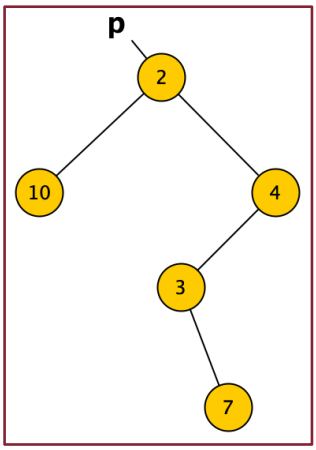
Tiziana Calamoneri

SAPIENZA
UNIVERSITÀ DI ROMA

Slides realizzate sulla base di quelle preparate da T. Calamoneri e G. Bongiovanni per il corso di Informatica Generale tenuto a distanza nell'A.A. 2019/20

Esercizio svolto 1 (1)

Esercizio 1. Progettare una funzione Crea che, dato il puntatore alla radice di un albero binario memorizzato tramite puntatori, restituisca l'albero in notazione posizionale.



0	1	2	3	4	5	6	7	8	9	10	11	12
2	10	4	--	--	3	--	--	--	--	--	--	7

Esercizio svolto 1 (2)

Soluzione.

Se l'albero puntato da p contiene nodi, allora si quantifica lo spazio necessario per inserire nell'array *A* i nodi dell'albero (già sappiamo che questo valore può essere esponenziale in n…) e poi si inseriscono in *A* i nodi dell'albero nelle posizioni opportune:

```
def crea(p):
    if p == None:
        return []
    n = spazio(p)
    A = [None]*(n+1)
    inserisci(p, A)
    return A
```


Esercizio svolto 1 (3)

```
def spazio(p, x = 0):
    # restituisce la locazione massima
    # necessaria a sistemare i nodi nella
    # versione posizionale dell'albero non vuoto
    a = b = 0
    if p.left:
        a=spazio(p.left, 2*x+1)
    if p.right:
        b=spazio(p.right, 2*x+2)
    return max(a,b,x)
```

Esercizio svolto 1 (4)

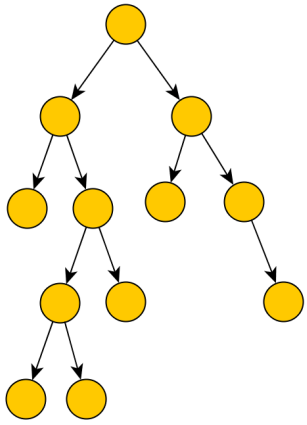
```
def inserisci(p, A, x=0):
    # per ogni nodo dell'albero a puntatori
    # crea un nodo  nell'array A
    if p!= None:
        A[x]=p.key
        if p.left:
            inserisci(p.left, A, 2*x+1)
        if p.right:
            inserisci(p.right, A, 2*x+2)
```


Esercizio svolto 2* (1)

Esercizio 2. Dato il puntatore alla radice di un albero binario non vuoto di n nodi, memorizzato tramite record e puntatori, progettare un algoritmo che restituisca il livello minimo in cui compaiono le sue foglie.

L'algoritmo deve avere costo computazionale O(n).

Esempio: per questo albero binario la risposta deve essere 2



Esercizio svolto 2 (2)

Soluzione: Eseguiamo una visita in post-ordine, perché ogni nodo restituisce al padre informazioni sulle foglie del suo sottoalbero:

```
def es(p):
    if p.left == p.right == None: return 0
    if p.left == None: return es(p.right) + 1
    if p.right == None: return es(p.left) + 1
    return min(es(p.right), es(p.right)) +1
```

Il costo è quello di una visita, e quindi $\Theta(n)$.

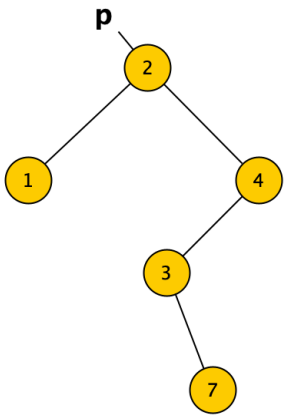
Esercizio svolto 3* (1)

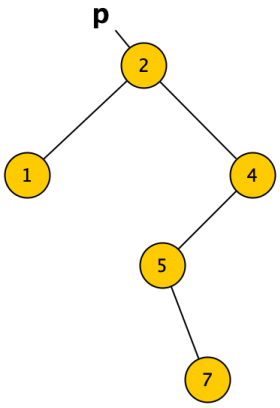
Esercizio 3. Progettare una funzione che, dato il puntatore alla radice di un albero binario non vuoto memorizzato tramite record e puntatori, verifichi se nell'albero è presente un cammino radice-foglia la cui sequenza di chiavi incontrate sia strettamente crescente. L'algoritmo deve avere costo computazionale $O(n)$ dove n è il numero di nodi dell'albero.

…

Esercizio svolto 3 (2)

Esempio: per l'albero a sinistra la risposta deve essere *False* mentre per l'albero di destra la risposta deve essere *True* (grazie al cammino 2, 4, 5, 7).





Esercizio svolto 3 (3)

OSSERVZIONE:
partendo dalla radice, non possiamo sapere se l'ipotetico cammino cercato sia quello di destra o quello di sinistra, ma se partiamo dalle foglie la verifica risulta più semplice, perché ogni nodo ha un unico padre…
dunque, anche in questo caso, effettuiamo una visita in post-ordine, in cui ogni nodo restituisce *True* se, risalendo al padre, il cammino percorso dalla foglia ha le proprietà richieste, e *False* altrimenti:

Esercizio svolto 3 (4)

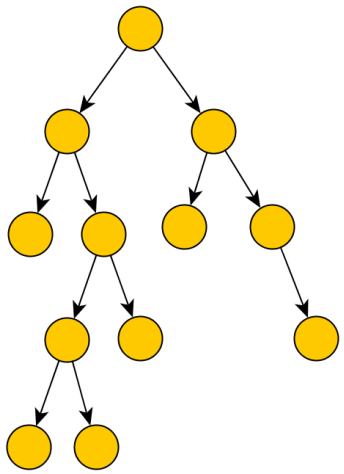
```
def es(p):
    if p.left == p.right == None:
        return True
    if p.left!=None and es(p.left) and p.key <p.left.key:
        return True
    return p.right!=None and es(p.right) and p.key <p.right.key
```

Il costo è quello di una visita.

Esercizio svolto 4* (1)

Esercizio 4. In un albero binario sono presenti nodi con un solo figlio, nodi con due figli e nodi senza figli (le foglie).
Progettare un algoritmo che, dato il puntatore alla radice di un albero binario di n nodi, restituisca la terna di interi (x,y,z) dove x è il numero di nodi con due figli, y il numero di nodi con 1 figlio e z il numero di foglie.
L'algoritmo deve avere costo computazionale O(n).
…

Esercizio svolto 4 (2)



Ad esempio per l'albero binario in figura l'algoritmo deve restituire la terna (5, 1, 6).

Esercizio svolto 4 (3)

Di nuovo, è necessario che ciascun nodo riceva informazioni da entrambi i suoi figli e poi le rielabori per inviarle a suo padre.
Per questo, la funzione seguirà una filosofia in post-ordine.
Ogni nodo restituisce al padre la terna corrispondente al proprio sottoalbero:

- le foglie restituiscono la terna (0,0,1);
- i nodi interni ricavano una terna da ciascuno dei figli, le sommano in una nuova terna incrementando di 1 la componente corrispondente alla sua situazione. Restituisce infine al padre la terna così calcolata.

Esercizio svolto 4 (4)

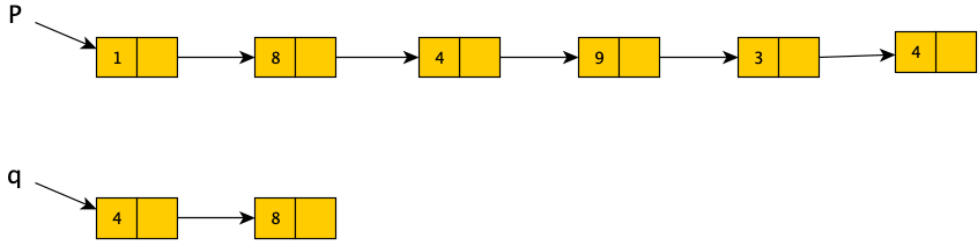
```
def es(p):
    if p== None:
        return 0,0,0
    if p.left == p.right == None:
        return 0, 0, 1
    if p.left != None:
        dueS, unoS, zeroS = es(p.left)
    if p.right != None:
        dueD, unoD, zeroD = es(p.right)
    if p.left == None:
        return dueD, unoD + 1, zeroD
    if p.right == None:
        return dueS, unoS + 1, zeroS
    return dueS + dueD + 1, unoS + unoD, zeroS + zeroD
```


Esercizio svolto 5 (1)

- Esercizio 5. Progettare una funzione che prende la testa di una lista puntata di n nodi e restituisce il puntatore ad una nuova lista puntata che contiene i nodi con le chiavi pari presenti nella prima lista.
- La lista originaria non va modificata, le chiavi della nuova lista devono essere ordinate in modo crescente e la lista non deve contenere duplicati.
- Il costo dell'algoritmo deve essere $O(n^2)$.
- …

Esercizio svolto 5 (2)

Per esempio per la lista p va restituita la lista q



- Descrivere l'algoritmo, scrivere lo pseudocodice e dimostrare il costo computazionale

Esercizio svolto 5 (3)

IDEA:

- L'algoritmo parte con la lista q vuota.
- Scorre la lista p e, per ogni chiave pari x che incontra, la inserisce in q (se non già presente) in modo ordinato tramite una funzione Inserisci(q, x).

L'idea sembra corretta. Il costo computazionale è $O(n^2)$ come richiesto?

Esercizio svolto 5 (4)

- La funzione Inserisci richiede tempo O(m) dove m ≤ n è la lunghezza di q.
  Infatti la funzione Inserisci scorre la lista ordinata q fino a che non trova x (e in quel caso termina) o trova il posto dove inserire x (e in quel caso lo inserisce).
- La funzione principale descritta richiede Θ(n) iterazioni per scorrere p e ad ogni iterazione c'è al più un'invocazione della funzione Inserisci.
- Quindi il costo totale rientra in $O(n^2)$.

Esercizio svolto 5 (5)

```
def es(p):
    q = None
    while p != None:
        if p.key%2 == 0:
            q = inserisci(q, p.key)
        p=p.next
    return q
```

```
def inserisci(q,x):
    if q == None or q.key > x:
        return Nodo(x,q)
    if q.key != x:
        q.next = inserisci(q.next, x)
    return q
```

Esercizio svolto 6 (1)

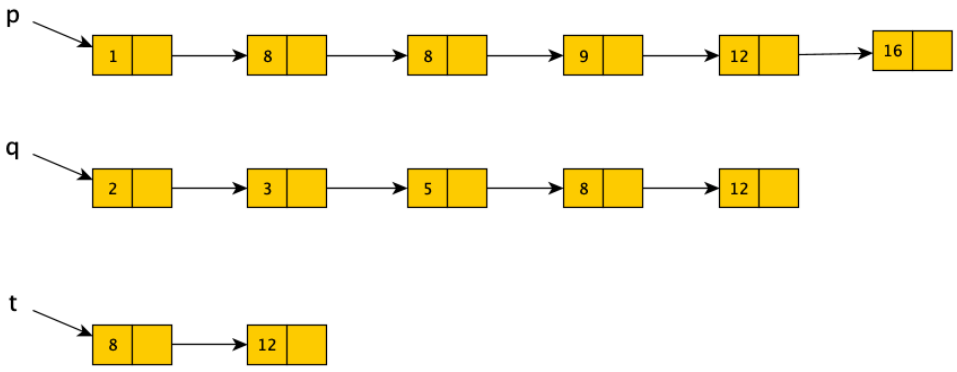
Esercizio 6. Siano date due liste puntate, ciascuna priva di duplicati, ordinate in modo non decrescente.
Progettare un algoritmo che, dati i puntatori p e q alle teste delle due liste, restituisca il puntatore alla testa di una terza lista contenente le chiavi comuni alle due liste. Non è importante l'ordine con cui le chiavi compaiono nella terza lista ma questa non deve contenere duplicati. Le due liste di partenza non vanno alterate.

La procedura deve avere costo O(n+m) dove n e m sono le lunghezze delle due liste.

...

Esercizio svolto 6 (2)

- Per esempio per le due liste p e q la lista da restituire è la lista t:



- Descrivere l'algoritmo, scrivere lo pseudocodice e dimostrare il costo computazionale.

Esercizio svolto 6 (3)

IDEA:

Possiamo implementare una modifica della funzione Fondi del MergeSort:

- Si scorrono le due liste tramite p e q a partire dal loro primo elemento.
- Se i due puntatori puntano alla stessa chiave, questa viene inserita in testa alla lista t ed entrambi i puntatori scorrono
- Se, al contrario, i due puntatori puntano a chiavi distinte, la minore non può trovarsi anche nell'altra lista ed il puntatore corrispondente può scorrere
- Appena uno dei due puntatori arriva a fine lista, la procedura termina restituendo t.

Esercizio svolto 6 (4)

L'idea è corretta. Rispetta il limite di costo richiesto?

- Ad ogni iterazione almeno una dei due puntatori viene incrementato quindi la procedura termina dopo al più O(n + m) iterazioni.
- Ogni iterazione è costante, perché si tratta di verificare la relazione tra le due chiavi puntate da p e q.

Esercizio svolto 6 (5)

```
def es(p,q):
    t=None
    while p!=None and q!= None:
        if p.key == q.key:
            t=Nodo(p.key,t)
            while p.next!= None and p.next==p.key: p=p.next
            p=p.next
            while q.next!= None and q.next==q.key: q=q.next
            q=q.next
        elif p.key<q.key:
            while p.next!= None and p.next==p.key: p=p.next
            p=p.next
        else:
            while q.next!= None and q.next==q.key: q=q.next
            q=q.next
    return t
```


Esercizio svolto 7 (1)

Esercizio 7. Sia dato un ABR con n nodi memorizzato tramite record e puntatori left e right (non c'è il puntatore al padre).

Progettare una procedura ricorsiva che, dato il puntatore alla radice ed un valore k ≥ 1, restituisca il valore del nodo che avrebbe posizione k nell'elenco delle chiavi ordinate.

Si deve restituire None nel caso l'albero abbia meno di k nodi.

L'algoritmo deve avere costo computazionale O(n).

Esercizio svolto 7 (2)

Idea 1.

- Ricordando che la visita in-oreder su un ABR produce la sequenza ordinata, si effettua tale visita
- si inseriscono le chiavi via via incontrate in un array A
- se A ha almeno k elementi, si restituisce il valore $A[k - 1]$.

Questo algoritmo ha costo $\Theta(n)$ ed utilizza un array d'appoggio, quindi memoria aggiuntiva dell'ordine di $\Theta(n)$.

Possiamo fare meglio di così?

Esercizio svolto 7 (3)

Idea 2.

- Per risparmiare la memoria aggiuntiva, possiamo contare gli elementi anziché salvare in un array.
- Possiamo fermarci dopo aver stampato il k-esimo elemento

Il costo computazionale è $O(k)$.

Esercizio svolto 7 (4)

```
def es(p,k, i=0):
    if i == k: return k
    if p == None:
        return i
    i = es(p.left,k, i )
    if i == k: return k
    if i == k-1:
        print(p.key)
        return k
    return es(p.right,k,i+1)
```


Esercizio svolto 8* (1) (esame giu. '21)

- Esercizio 8. Data una sequenza S di n bit memorizzata in un array A, progettare un algoritmo che ordini S in tempo $\Theta(n)$.
- L'algoritmo deve ordinare in loco.
- Dell'algoritmo proposto si dia la descrizione a parole, si scriva lo pseudocodice e si calcoli il costo computazionale.
- Per quale motivo è possibile ordinare S in tempo lineare?

Esercizio svolto 8 (2)

- IDEA 1. si può usare il Counting sort senza dati satellite, visto che l'array A contiene solo due valori, 0 e 1.
- Si ottiene un algoritmo lineare perché l'array contiene un numero costante, cioè O(n), di valori.

Esercizio svolto 8 (3)

- IDEA 2. Alternativamente, è possibile implementare una leggera modifica della funzione Partiziona del Quicksort, per spostare a sinistra tutti gli zeri ed a destra tutti gli uni.
- Anche in questo caso l'algoritmo è lineare come il costo di Partiziona

Esercizio svolto 9* (1) (esame lug. '21)

Esercizio 9. Progettare un algoritmo che, dato un array A di n interi distinti i cui elementi sono all'inizio in ordine crescente e da un certo punto in poi in ordine decrescente, restituisce in tempo O(log n) il massimo intero presente nell'array. Ad esempio: per A = [8,10,20,80,100,200,400,500,3,2,1] l'algoritmo deve restituire il valore 500.

Esercizio svolto 9 (2)

Soluzione.

Se l'array contiene 1 o 2 elementi risolvo il problema direttamente. In caso contrario applico la ricerca binaria: sia m la posizione di mezzo dell'array, se A[m - 1] < A[m] cerco nel sottoarray di destra che parte dalla posizione m in caso contrario cerco nel sottoarray di sinistra che termina nella posizione A[m - 1].

Esercizio svolto 9 (3)

```
def esame2(A):
    i=0
    j = len(A) - 1
    while True:
        if j == i: return A[i]
        if j == i + 1: return max(A[i], A[i + 1])
        m = (i + j)//2
        if A[m-1]<A[m]:i=m
        else: j=m-1
```

Costo: O(log n)

Esercizio svolto 10* (1) (esame lug. '21)

Esercizio 10. Dato un albero binario T, radicato e con n nodi, definiamo un nodo u di T equilibrato se il sottoalbero sinistro di u e il sottoalbero destro di u hanno entrambi lo stesso numero di nodi. Progettare un algoritmo che, dato il puntatore r alla radice di un albero binario memorizzato tramite record e puntatori, restituisca in tempo O(n) il numero dei suoi nodi equilibrati.

Esercizio svolto 10 (2)

Soluzione. si effettua una visita postorder dell'albero. Ogni nodo restituisce al padre:

- il numero di nodi presenti nel suo sottoalbero
- il numero di nodi equilibrati presenti nel suo sottoalbero.

Le foglie restituiscono i valori 0 e 0.

Esercizio svolto 10 (3)

... Ciascun nodo interno, ricevendo queste informazioni, è in grado di trasmettere a sua volta le giuste informazioni al padre:

- aggiungendo 1 alla somma dei nodi del sottoalbero sinistro e del sottoalbero destro può calcolare il numero di nodi del suo sottoalbero
- il numero di nodi equilibrati nel suo sottoalb. è la somma dei nodi equilibrati del sottoalb. sin. + nodi equilibrati del sottoalb. des. + eventualmente 1 (se il nodo stesso è equilibrato).

Esercizio svolto 10 (4)

… Il numero di nodi equilibrati restituito dalla radice è la soluzione al problema.

```
def esame3(r):
   if r==None:
      return 0, 0
equilibratiS, nodiS =esame3(r.left)
equilibratiD, nodiD =esame3(r.right)
equilibrati = equilibratiS + equilibratiD
if nodiS == nodiD: equilibrati+ = 1
return equilibrati, nodiS + nodiD + 1
```


Esercizio svolto 11* (1) (esame giu. '22)

Esercizio 11. Sia A un array di n interi positivi. Con la coppia ordinata (i,j), $0 \le i \le j < n$, rappresentiamo il suo sottoarray che parte dall'elemento in posizione i e termina con l'elemento in posizione j, definiamo *valore* di un sottoarray come la somma dei suoi elementi.
Progettare un algoritmo che, dato A ed un intero positivo s, restituisce la coppia ordinata che rappresenta il sottoarray di A più a sinistra che ha valore s. Se un tale sottoarray non esiste, la funzione deve restituire None. L'algoritmo deve avere costo computazionale O(n).

Esercizio svolto 11 (2)

Ad esempio, per A = [1,3,5,2,9,3,3,1,6] con s = 7, l'algoritmo deve restituire la coppia (2, 3) (ci sono infatti in A tre sottoarray con valore 7 le cui coppie nell'ordine da sinistra a destra sono (2, 3), (5, 7), (7, 8)).
Con s = 21 l'algoritmo deve restituire None in quanto A non ha sottoarray con valore 21.

Esercizio svolto 11 (3)

Soluzione.

Consideriamo i vari sottoarray di A utilizzando:

- due indici i e j al primo e all'ultimo elemento del sottoarray
- una variabile tot con il valore del sottoarray.

Inizializziamo i due indici a 0 e di conseguenza tot ad A[0].

Incrementiamo di volta in volta gli indici distinguendo 3 possibili casi:

Esercizio svolto 11 (4)

…

1. il valore del sottoarray in esame è inferiore ad s: bisognerà incrementare j aggiungendo un altro elemento al sottoarray.
   Il valore del nuovo array sarà tot + A[j].
2. il valore del sottoarray in esame è s: abbiamo trovato il sottoarray e restituiamo i due indici i e j.
3. il valore del sottoarray è superiore ad s: dobbiamo incrementare i escludendo il primo elemento dal sottoarray.
   Il valore del nuovo array sarà tot - A[i - 1]

Esercizio svolto 11 (5)

…

Ovviamente con le varie operazione di incremento di i e j dobbiamo far attenzione che si abbia sempre i ≤ j e j < n.

```
def es2(A,s):
    i=j=tot=0
    while j<len(A):
        print(i,j)
        tot+=A[j]
        while tot>s:
            tot-=A[i]
            i+=1
        if tot==s: return i,j
        j+=1
    return None
```

il costo dipende dal numero di iterazioni dei while. Ad ogni iterazione del while più interno si incrementa l'indice i e ad ogni iterazione del while più esterno si incrementa l'indice j; quindi il numero di iterazioni è Θ(n).