Laboratorio 10

Programmazione - CdS Matematica

Michele Donini
14 Gennaio 2014

Esercizio: riepilogo.

- Generare 100000 numeri reali in [0,1[e scriverli su un file.
- Leggere dal file i valori e generare un albero bst (A) dei valori letti.
- Creare una funzione che trova, se esiste, un valore in A contenuto in un intervallo specifico passato come argomento (ritorna il primo che trova).
- Verificare il numero di chiamate ricorsive medio necessario per la ricerca di un valore contenuto in un intervallo (eseguendo più volte la funzione precedente, per intervalli diversi).

Parte 1 - Scrittura su File

- Generare 100000 numeri reali in [0,1[e scriverli su un file, uno per riga.

Parte 1 - Scrittura su File

- Generare 100000 numeri reali in [0,1[e scriverli su un file, uno per riga.

```
def genera_valori(n):
    filevalori = open("valori","w")
    for v in range(n):
        val = random.random()
        filevalori.write(str(val)+'\n')
    filevalori.close()
```

Parte 2 - Lettura da File

- Leggere e stampare i valori contenuti nel file generato nel punto precedente.

Parte 2 - Lettura da File

- Leggere e stampare i valori contenuti nel file generato nel punto precedente.

```
filevalori = open("valori","r")
for v in filevalori.readlines():
    val = float(v)
    print val
```

Parte 2 - Generazione BST

- Leggere dal file i primi 100 valori e generare un albero bst (A) dei valori letti.

Parte 2 - Generazione BST

- Leggere dal file i primi 100 valori e generare un albero bst (A) dei valori letti.

```
class Albero:
    def __init__(self,val=None,sx=None,dx=None):
        self.val = val
        self.sx = sx
        self.dx = dx
```

Parte 2 - Generazione BST

- Leggere dal file i primi 100 valori e generare un albero bst (A) dei valori letti.

```
class Albero:
    def __init__(self,val=None,sx=None,dx=None):
        self.val = val
        self.sx = sx
        self.dx = dx
```

```
def bst_ins(albero,v):
    if not albero:
        return Albero(v)
    if albero.val>v:
        albero.sx = bst_ins(albero.sx,v)
    if albero.val<v:
        albero.dx = bst_ins(albero.dx,v)
    return albero
```

Parte 2 - Generazione BST

Creiamo quindi l'albero richiesto, iniziando con i primi 100 valori contenuti nel file (100 nodi).

Parte 2 - Generazione BST

Creiamo quindi l'albero richiesto, iniziando con i primi 100 valori contenuti nel file (100 nodi).

```
numero_nodi = 100
filevalori = open("valori","r")
A = None
lista_valori  = filevalori.readlines()
for v in range(numero_nodi):
    val = float(lista_valori[v])
    A = bst_ins(A,val)
```

Ripasso 1

- Definire la funzione di ricerca di un elemento in un albero bst (se l'elemento cercato esiste ritorna il sottoalbero che lo ha come radice, altrimenti ritorna None).

Ripasso 1

- Definire la funzione di ricerca di un elemento in un albero bst (se l'elemento cercato esiste ritorna il sottoalbero che lo ha come radice, altrimenti ritorna None).

```
def bst_search(albero, v):
    if not albero:
        return None
    if albero.val==v:
        return albero
    if albero.val>v:
        return bst_search(albero.sx,v)
    return bst_search(albero.dx,v)
```

Ripasso 2

- Definire una funzione che stampa a video la lista ordinata di elementi contenuta in un corretto albero bst.

Ripasso 2

- Definire una funzione che stampa a video la lista ordinata di elementi contenuta in un corretto albero bst.

```
def print_lista_albero(albero):
    if not albero:
        return
    if albero.sx:
        print_lista_albero(albero.sx)
    print albero.val
    if albero.dx:
        print_lista_albero(albero.dx)
```

Parte 3 - Funzione di ricerca

- Creare una funzione che trova, se esiste, un valore in A (albero bst) contenuto in un intervallo specifico passato come argomento (ritorna il primo elemento che trova nell'intervallo o None se non ve ne sono).

Parte 3 - Funzione di ricerca

- Creare una funzione che trova, se esiste, un valore in A (albero bst) contenuto in un intervallo specifico passato come argomento (ritorna il primo elemento che trova nell'intervallo o None se non ve ne sono).

```
def bst_search_intervallo(albero, s, e):
    if not albero:
        return None
    if albero.val>=s and albero.val<=e:
        return albero.val
    if albero.val>e:
        return bst_search_intervallo(albero.sx,s,e)
    return bst_search_intervallo(albero.dx,s,e)
```

Parte 4 - Calcolo del numero di ricorsioni

- Verificare il numero di chiamate ricorsive necessario per la ricerca di un valore contenuto in un intervallo.
- Suggerimento: aggiungere una variabile globale che faccia da contatore delle chiamate ricorsive.

Parte 4 - Calcolo del numero di ricorsioni

- Verificare il numero di chiamate ricorsive necessario per la ricerca di un valore contenuto in un intervallo.
- Suggerimento: aggiungere una variabile globale che faccia da contatore delle chiamate ricorsive.

```
conta_iterazioni = ...
def bst_search_intervallo(albero, s, e):
    global conta_iterazioni
...
```

Parte 4 - Calcolo del numero di ricorsioni

- Verificare il numero di chiamate ricorsive necessario per la ricerca di un valore contenuto in un intervallo.
- Suggerimento: aggiungere una variabile globale che faccia da contatore delle chiamate ricorsive.

```
conta_iterazioni = 0
def bst_search_intervallo(albero, s, e):
    global conta_iterazioni
    conta_iterazioni = conta_iterazioni + 1
    if not albero:
        return None
    if albero.val>=s and albero.val<=e:
        return albero.val
    if albero.val>e:
        return bst_search_intervallo(albero.sx,s,e)
    return bst_search_intervallo(albero.dx,s,e)
```

Parte 4 - Calcolo empirico della complessità

- Verificare il numero di chiamate ricorsive **medio** (replicando più volte il punto precedente e calcolando il valore medio) necessario per la ricerca di un valore contenuto in un intervallo di ampiezza fissa 0.00001 contenuto in [0,1].
- Modificare il numero di nodi caricati nell'albero.

Parte 4 - Calcolo empirico della complessità

- Verificare il numero di chiamate ricorsive **medio** (replicando più volte il punto precedente e calcolando il valore medio) necessario per la ricerca di un valore contenuto in un intervallo di ampiezza fissa 0.00001 contenuto in [0,1].
- Modificare il numero di nodi caricati nell'albero.

```
for i in range(10000):
    s = random.random()
    e = s + 0.00001
    if e > 1.0:
        e,s = 1.0,e - 0.00001
    out = bst_search_intervallo(A,s,e)
print conta_iterazioni/float(i+1)
```

Parte 4 - Calcolo empirico della complessità

- Verificare il numero di chiamate ricorsive **medio** (replicando più volte il punto precedente e calcolando il valore medio) necessario per la ricerca di un valore contenuto in un intervallo di ampiezza fissa 0.00001 contenuto in [0,1].
- Modificare il numero di nodi caricati nell'albero.

```
for i in range(10000):
    s = random.random()
    e = s + 0.00001
    if e > 1.0:
        e,s = 1.0,e - 0.00001
    out = bst_search_intervallo(A,s,e)
print conta_iterazioni/float(i+1)
```

Una possibile soluzione:

```
http://www.math.unipd.it/~mdonini/lab10.py
```