Laboratorio 8

Programmazione - CdS Matematica

Mirko Polato

15 Dicembre 2015

Operare su File

Per operare su un file occorre eseguire le seguenti operazioni:

- aprire il file in scrittura o lettura;
- scrivere o leggere su file (a seconda della modalità in cui è stato aperto)
- chiudere il file

```
# scrittura su file
outfile = open("test_file.txt","w")
outfile.write("Questo e' un testo di prova.\nProva ad
    aprire il file e guardare che cosa c'e' dentro.\n")
outfile.close()
```

```
# lettura da file
infile = open("test_file.txt","r")
testo = infile.read()
infile.close()
print testo
```

Altri metodi dell'oggetto file

- `readline()`: legge una sola riga
- `readlines()`: ritorna l'intero file come lista di righe
- `writelines(L)`: scrive in ogni nuova riga gli elementi della lista L

```
infile = open("test_file.txt","r")
l = infile.readlines()
infile.close()
print l
```

```
["Questo e' un testo di prova.\n", "Prova ad aprire il
    file e guardare che cosa c'e' dentro.\n"]
```

Scrittura su File

- Definire una funzione `genera_valori(N)` che generi N numeri interi in $[0,10^4[$ e li scriva su un file, uno per riga.

Scrittura su File

- Definire una funzione `genera_valori(N)` che generi N numeri interi in $[0,10^4[$ e li scriva su un file, uno per riga.

```
from random import randint
def genera_valori(N):
    filevalori = open("valori.txt","w")
    for v in range(N):
        val = randint(0, 10000)
        filevalori.write(str(val)+'\n')
    filevalori.close()

genera_valori(100000)
```

Lettura da File

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

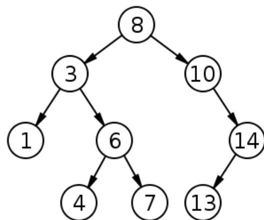
Lettura da File

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

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

Alberi binari di ricerca

Gli *alberi binari di ricerca* (binary search trees, **BST**), detti anche *alberi ordinati*, sono particolari alberi binari con la seguente proprietà:
il valore di ogni nodo è maggiore del valore di tutti i nodi del sotto-albero sinistro e minore del valore di tutti i nodi del sotto-albero destro.



Classe Albero

Definire la classe Albero (binario). È sufficiente definire il solo costruttore.

Classe Albero

Definire la classe Albero (binario). È sufficiente definire il solo costruttore.

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

Generazione BST

Definire la funzione `bst_ins(T,v)` (esterna alla classe Albero) che inserisca il valore v nell'albero T (anche vuoto) mantenendo la proprietà dei BST. La funzione deve ritornare l'albero con il nuovo valore inserito.

Generazione BST

Definire la funzione `bst_ins(T,v)` (esterna alla classe Albero) che inserisca il valore v nell'albero T (anche vuoto) mantenendo la proprietà dei BST. La funzione deve ritornare l'albero con il nuovo valore inserito.

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

Generazione BST

- Leggere dal file i primi 100 valori e generare un BST dei valori letti.

Generazione BST

- Leggere dal file i primi 100 valori e generare un BST dei valori letti.

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

Funzione di stampa

- Definire una funzione `print_lista_albero(T)` che stampa a video gli elementi contenuti in un albero BST ordinati in modo crescente.

Funzione di stampa

- Definire una funzione `print_lista_albero(T)` che stampa a video gli elementi contenuti in un albero BST ordinati in modo crescente.

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

Funzione di ricerca (valore esatto)

- Definire la funzione di ricerca `bst_search(T, v)` di un elemento v in un BST T: se l'elemento cercato esiste ritorna il sotto-albero che ha v come radice, altrimenti ritorna None.

Funzione di ricerca (valore esatto)

- Definire la funzione di ricerca `bst_search(T, v)` di un elemento v in un BST T: se l'elemento cercato esiste ritorna il sotto-albero che ha v come radice, altrimenti ritorna None.

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

Funzione di ricerca (intervallo)

- Creare una funzione `bst_search_intervallo(T, a, b)` che trova, se esiste, un valore in T contenuto in un intervallo chiuso [a,b] (ritorna il primo elemento appartenente all'intervallo o None se non ve ne sono).

Funzione di ricerca (intervallo)

- Creare una funzione `bst_search_intervallo(T, a, b)` che trova, se esiste, un valore in T contenuto in un intervallo chiuso [a,b] (ritorna il primo elemento appartenente all'intervallo o None se non ve ne sono).

```
def bst_search_intervallo(T, a, b):
    if not T:
        return None
    if T.val>=a and T.val<=b:
        return T.val
    if T.val>b:
        return bst_search_intervallo(T.sx,a,b)
    return bst_search_intervallo(T.dx,a,b)
```

Calcolo del numero di ricorsioni

- Calcolare il numero di chiamate ricorsive necessario per la ricerca di un valore contenuto in un intervallo. Suggerimento: definire una variabile globale che faccia da contatore delle chiamate ricorsive alla funzione `bst_search_intervallo`.

```
conta_iterazioni = ...
def bst_search_intervallo(T, a, b):
    global conta_iterazioni
    ...
```

Calcolo del numero di ricorsioni

- Calcolare il numero di chiamate ricorsive necessario per la ricerca di un valore contenuto in un intervallo. Suggerimento: definire una variabile globale che faccia da contatore delle chiamate ricorsive alla funzione `bst_search_intervallo`.

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

Calcolo empirico della complessità

- Calcolare il numero di chiamate ricorsive **medio** necessario per la ricerca di un valore in un intervallo di ampiezza fissa 10 contenuto in $[0,10^4[$: replicare più volte ($\approx 10^4$) il calcolo precedente variando ogni volta l'intervallo in modo casuale.
- Ripetere il calcolo modificando il numero di nodi caricati nell'albero (n_nodi = 100, 1000, 10000, ...).

Calcolo empirico della complessità

- Calcolare il numero di chiamate ricorsive **medio** necessario per la ricerca di un valore in un intervallo di ampiezza fissa 10 contenuto in $[0{,}10^4[$: replicare più volte ($\approx 10^4$) il calcolo precedente variando ogni volta l'intervallo in modo casuale.
- Ripetere il calcolo modificando il numero di nodi caricati nell'albero ($n_nodi = 100, 1000, 10000, \ldots$).

```
conta_iterazioni = 0
n_prove = 10000
for i in range(n_prove):
    a = randint(0, 10000 - 10)
    b = a + 10
    out = bst_search_intervallo(A,a,b)
print conta_iterazioni/float(n_prove)
```