Laboratorio di Algoritmi e Strutture Dati

Esercitazioni del 22 Gennaio 2013

Esercizio 2: stringsearchtree, treepair

L'esercizio si compone di due parti. Entrambe prevedono la modifica del codice fornito in `searchtree.c` (e `searchtree.h`) per l'implementazione di alberi binari di ricerca su chiavi di tipo `int`.

Prima parte: stringsearchtree

Modificare l'implementazione di `searchtree` in modo da ottenere un'implementazione per alberi binari di ricerca in cui l'insieme delle possibili chiavi è costituito dalle stringhe di caratteri, con l'ordinamento lessicografico (vale a dire, l'ordinamento del dizionario).

Il programma di test deve poter essere lanciato come:

`teststringsearchtree` n $file$

dove n è un intero positivo e $file$ è il nome di un file contenente almeno n *parole* —dove per *parola* intendiamo una sequenza consecutiva di caratteri limitata da caratteri di spaziatura (ad esempio provare il programma con il file `testdata.txt`).

Il programma deve:

1. Leggere n parole da $file$ e inserirle in un albero binario di ricerca per stringhe.
2. Stampare l'elenco ordinato delle parole contenute nell'albero.
3. Chiedere all'utente di immettere una parola da cancellare.
4. Cancellare, se essa occorre, la parola scelta dall'albero.
5. Stampare l'elenco ordinato delle parole contenute nell'albero.

Seconda parte: treepair

Modificare l'implementazione di `searchtree` e di `stringsearchtree` per realizzare una struttura dati combinata che supporti la ricerca sia su interi che su stringhe.

In particolare, si definiscano le seguenti strutture:

```
typedef struct treepairnode treepairnode;

typedef struct {
    int intero;
    char *stringa;
    treepairnode *i, *s;
} datanode;

struct treepairnode {
    datanode *data;
```

```
    treepairnode *left, *right, *up;
};

typedef struct {
    treepairnode *iroot, *sroot;
} treepair;
```

e le funzioni il cui prototipo è descritto in `treepair.h` in modo tale da implementare un `treepair`:

Per `treepair` intendiamo la combinazione di un albero di ricerca con chiavi intere con un albero di ricerca con chiavi stringhe, in modo tale da supportare la ricerca di dati costituiti da coppie (intero, stringa) sia attraverso il loro campo intero sia attraverso il loro campo stringa.

- Nella nostra implementazione, l'intera struttura del `treepair` è acceduta e manipolata attraverso un oggetto contenente le radici `iroot` e `sroot` di due distinti alberi binari di ricerca.
  La funzione `treepair *createtreepair()` deve dunque allocare un elemento di tipo `treepair` e inizializzare a `NULL` le due radici.
- Ogni nodo di ognuno dei due alberi è di tipo `treepairnode`, e contiene gli usuali puntatori `left, right, up`, congiuntamente a un puntatore `data` a un oggetto di tipo `datanode`.
- Un `datanode` contiene la coppia di valori `int intero` e `char *stringa`, congiuntamente a due puntatori `i` e `s` a `treepairnode`.
- Ogni `datanode` d punta attraverso il suo puntatore `i` a un nodo nell'albero binario con radice `iroot`, che a sua volta punta a d tramite il suo campo `data`.
  Analogamente, d punta attraverso il suo puntatore `s` a un nodo nell'albero binario con radice `sroot`, che a sua volta punta a d tramite il suo campo `data`.
  In questo modo si crea un'associazione bidirezionale tra un `treepairnode` nell'albero di radice `iroot`, un `datanode` e un `treepairnode` nell'albero di radice `sroot`.
- L'albero di ricerca con radice `iroot` utilizza come chiave il campo `intero` contenuto nei datanode puntati dai nodi dell'albero.
  Analogamente, l'albero di ricerca con radice `sroot` utilizza come chiave il campo `stringa` contenuto nei datanode puntati dai nodi dell'albero.

Esempio: Il `treepair` ottenuto inserendo nell'ordine le coppie:

```
(5,"pippo"),(2,"pluto"),(4,"topolino"),(1,"ziopaperone"),(7,"paperino")
```

è rappresentato in Figura 1.

Per quanto riguarda le funzioni da implementare:

- Le funzioni `inorder`, `treemin`, `treemax`, `treepred`, `treesucc` sono semplici adattamenti al tipo `treepairnode` delle analoghe funzioni del pacchetto `searchtree`.
- Le funzioni `getintroot`, `getstrroot`, `getintdatum`, `getstrdatum` sono semplici funzioni di una riga che restituiscono l'opportuno campo della struttura passata come argomento.
- Le funzioni `intsearch`, `strsearch` si ottengono adattando ai tipi la funzione `search` contenuta nel pacchetto `searchtree` e richiamando le due funzioni rispettivamente sulle radici `iroot` e `sroot` dei due alberi del `treepair`.

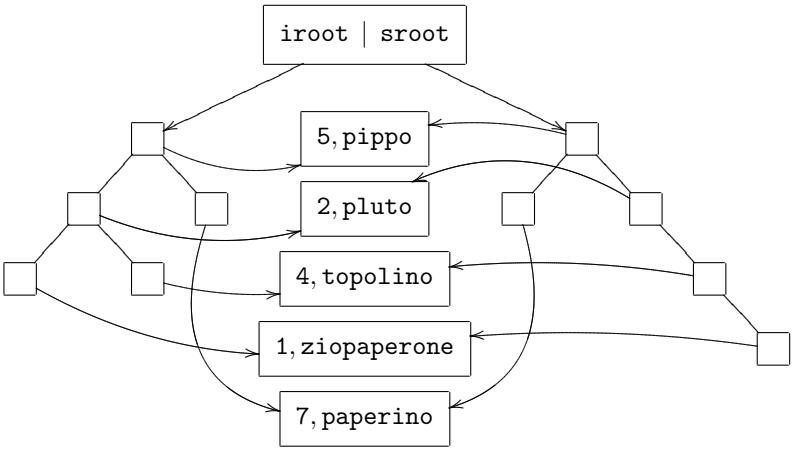

Figura 1:

- La funzione `insert` si ottiene:
  - creando due `treepairnode` e un `datanode` e stabilendo gli opportuni legami bidirezionali tra il `datanode` contenente i dati da inserire e i due `treepairnode`.
  - inserendo nel `searchtree` con radice `iroot` il `treepairnode` puntato dal campo `i` del `datanode` e nel `searchtree` con radice `sroot` il `treepairnode` puntato dal campo `s`.
- La funzione `delete` si ottiene in maniera analoga, ma c'è un punto piuttosto delicato da trattare.
  - `delete(q)` in primo luogo deve ottenere l'indirizzo del `treepairnode` corrispondente a `q` nell'altro albero: se `q` giace nell'albero di radice `iroot`, allora il `treepairnode` corrispondente è `q->data->s`.
    Si ragiona in maniera analoga nell'altro caso.
  - Si cancellano i dati dall'uno e dall'altro albero avendo cura di liberare il `datanode` puntato da `q` una volta sola.
  - Il punto delicato: nel caso in cui da uno (o da entrambi) gli alberi si cancelli il successore del nodo previo scambio dei dati, bisogna avere cura di ristabilire i legami giusti tra i `treepairnode` dei due alberi e il `datanode` associato: se `q` giace nell'albero di radice `iroot` e `r` è il successore da cancellare al posto di `q`, allora oltre a salvare il puntatore `data` di `r` con `q->data = r->data`, bisogna anche ricollegare il nodo con il partner di `r` nell'albero di radice `sroot`: questo si ottiene con `r->data->i = q`.
    Si ragiona analogamente se `q` giace nell'albero di radice `sroot`.

Il programma di test deve poter essere lanciato come:

`testtreepair` n $file$

dove n è un intero positivo e $file$ è il nome di un file contenente almeno n *parole* —dove per *parola* intendiamo una sequenza consecutiva di caratteri limitata da caratteri di spaziatura (ad esempio provare il programma con il file `testdata.txt`).

Il programma deve:

1. Leggere n parole $\alpha_1, \ldots, \alpha_n$ da $file$, calcolare la lunghezza l_i di α_i per ogni $i = 1, \ldots, n$ e inserire ogni coppia (l_i, α_i) in un `treepair`.
2. Stampare l'elenco delle coppie contenute nel `treepair` ordinato rispetto alle lunghezze.
3. Stampare l'elenco delle coppie contenute nel `treepair` ordinato rispetto alle parole.
4. Chiedere all'utente di immettere una parola da cancellare.
5. Cancellare dal `treepair`, se essa occorre, la coppia contenente la parola scelta.
6. Stampare l'elenco delle coppie contenute nel `treepair` ordinato rispetto alle lunghezze.
7. Stampare l'elenco delle coppie contenute nel `treepair` ordinato rispetto alle parole.
8. Chiedere all'utente di immettere la lunghezza delle parole da cancellare.
9. Cancellare dal `treepair` ogni coppia contenente una parola della lunghezza scelta.
10. Stampare l'elenco delle coppie contenute nel `treepair` ordinato rispetto alle lunghezze.
11. Stampare l'elenco delle coppie contenute nel `treepair` ordinato rispetto alle parole.