Facoltà di Scienze Matematiche Fisiche Naturali

Laboratorio di Algoritmi e Strutture Dati

Prof. Aniello Murano

Alberi Binari di Ricerca

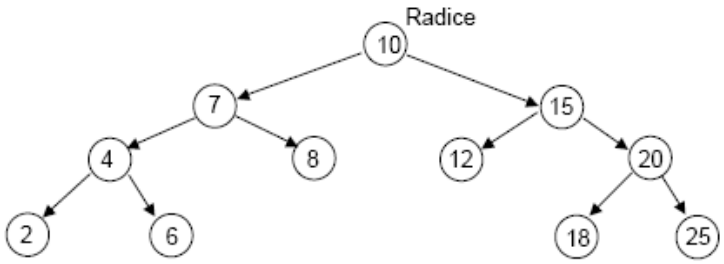
Cancellazione di un nodo

Corso di Laurea	Informatica	Lezione numero: 13
Codice insegnamento	13917	
Email docente	murano@na.infn.it	Parole chiave: Alberi Binari, Ricerca Binaria, Visite di Alberi
Anno accademico	2007/2008	

next

Alberi binari di ricerca (ABR)

- Un albero binario di ricerca (ABR) è un albero binario in cui per ogni nodo dell'albero N tutti i nodi del sottoalbero sinistro di N hanno un valore minore o uguale di quello di N e tutti i nodi del sottoalbero destro hanno un valore maggiore di quello del nodo N.



back next

Riepilogo della precedente lezione in aula

Nella lezione precedente abbiamo studiato come:

- Definire un ABR tramite struct
- Creare un ABR con un unico nodo o come unione tramite un nuovo nodo radice di due ABR preesistenti;
- Controllare se un ABR è vuoto;
- Controllare che un albero è un ABR;
- Ottenere il valore della radice di un albero;
- Avere il puntatore al figlio sx/dx di un ABR.
- Stampare il contenuto di un albero tramite una visita
- Ricercare un dato elemento o il minimo di un ABR

back next

Sommario

In questa lezione valuteremo come cancellare un nodo da un ABR.

back next

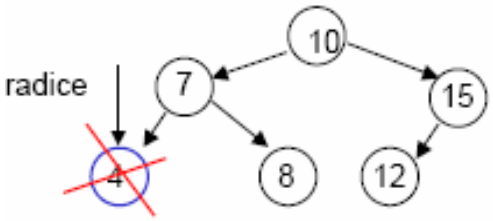
Cancellazione

Nella cancellazione di un elemento *el* da un albero bisogna distinguere i seguenti casi:

1. Albero vuoto: non viene realizzata alcuna cancellazione;
2. L'elemento el < radice albero: la cancellazione va effettuata nel sottoalbero sinistro;
   elimina(radice->sinistro,el);
3. L'elemento el > radice albero: la cancellazione va effettuata nel sottoalbero destro;
   elimina(radice->destro,el);
4. L'elemento el==radice albero. Si considerano i seguenti casi:
   1. La radice è una foglia:
   2. Il nodo ha un sottoalbero non vuoto e l'altro vuoto.
   3. Il nodo ha entrambi i sottoalberi non vuoti.

Primo caso

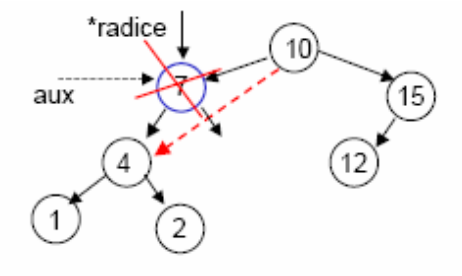
La radice è una foglia: viene eliminata liberando la memoria del nodo radice



Secondo Caso

Se il sottoalbero destro e' vuoto:

- il nodo figlio del nodo padre (10) di el (7) diventa il nodo figlio di el (4);
- Viene eliminato il nodo el (7).

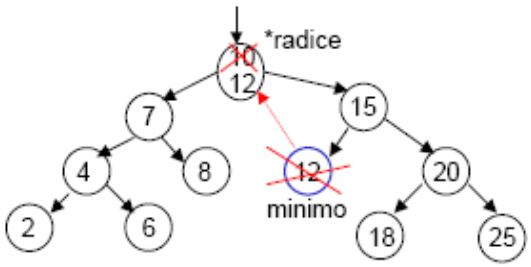


back next

Terzo Caso

Il sottoalbero destro e sinistro del nodo el sono non vuoti:

- viene ricavato il minimo (12) del sottoalbero destro;
- il minimo (12) viene copiato nel nodo el (10);
- viene eliminato il minimo (12) del sottoalbero destro.



back next

Codice per la cancellazione (NO)

```
void *elimina (struct nodo *radice, int el)
{   struct nodo *aux;
    if(vuoto(radice)) { printf("elemento non trovato"); return; }
    if(radice->inforadice > el) /* el va cercato nel sottoalbero sinistro */
          elimina(radice->sinistro,el);
    else if(radice->inforadice< el) /* el va cercato nel sottoalb. destro */
          elimina(radice->destro,el);
    else /* Trovato l'e'elemento da cancellare */
          {  if(radice->sinistro && radice->destro) /* Il nodo ha due figli*/
                    {         aux=ricerca_minimo(radice->destro);
                              radice->inforadice=aux->inforadice;
                              elimina(radice->destro,aux->inforadice); }
             else {  /* Il nodo ha 0 oppure un figlio*/
                    aux=radice;
                    if (radice->sinistro = NULL) radice=radice->destro;
                    else if (radice->destro = NULL) radice=radice->sinistro;
                    free(aux); }
          }
     return;
}
```

back next

Osservazioni

- Il codice precedente, non cancella efficacemente un nodo perchè all'atto della cancellazione "free(radice); radice=NULL;" deallochiamo effettivamente la memoria per il nodo da cancellare, ma il nodo padre di quest'ultimo continuerà a puntare ad una locazione di memoria reale, che non è NULL.
- La funzione è scorretta perchè porta troppo in avanti il puntatore da non permette la modifica il campo dx o sx del nodo padre.
- Una soluzione è quella di utilizzare una funzione che ritorni un puntatore a una struttura ad albero (come proposto per le liste). Il valore ritornato, opportunamente gestito nelle chiamate ricorsive, permetterà di modificare l'ABR nel modo voluto.
- Un'altra soluzione è quella di utilizzare il metodo del puntatore a puntatore. Questo è da preferirsi quando si ha a che fare con funzioni che devono modificare più oggetti.
- Di seguito mostriamo la seconda soluzione.

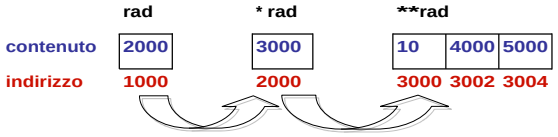
back next

Come usare il puntatore a puntatore

- Un ABR con doppio puntatore sarà definito nel seguente modo:

```
struct nodo  {int info; struct nodo *sx; struct nod *dx; }
struct nodo **rad;
rad=(struct nodo**)malloc(sizeof(struct nodo));
```

- L'ABR sarà referenziato con una cella contiene un indirizzo (rad) di una cella contenente un indirizzo (*rad) di una cella in cui è memorizzato la radice dell'albero. Si veda per es. la figura sotto



- Una chiamata ad una funzione "**elimina(**rad, el)**", con **el** intero da eliminare, passa alla funzione il riferimento alla cella con indirizzo 2000. Una modifica alla radice dell'ABR, modificherà anche il contenuto di questa cella.

back next

Uso di puntatori a puntatori

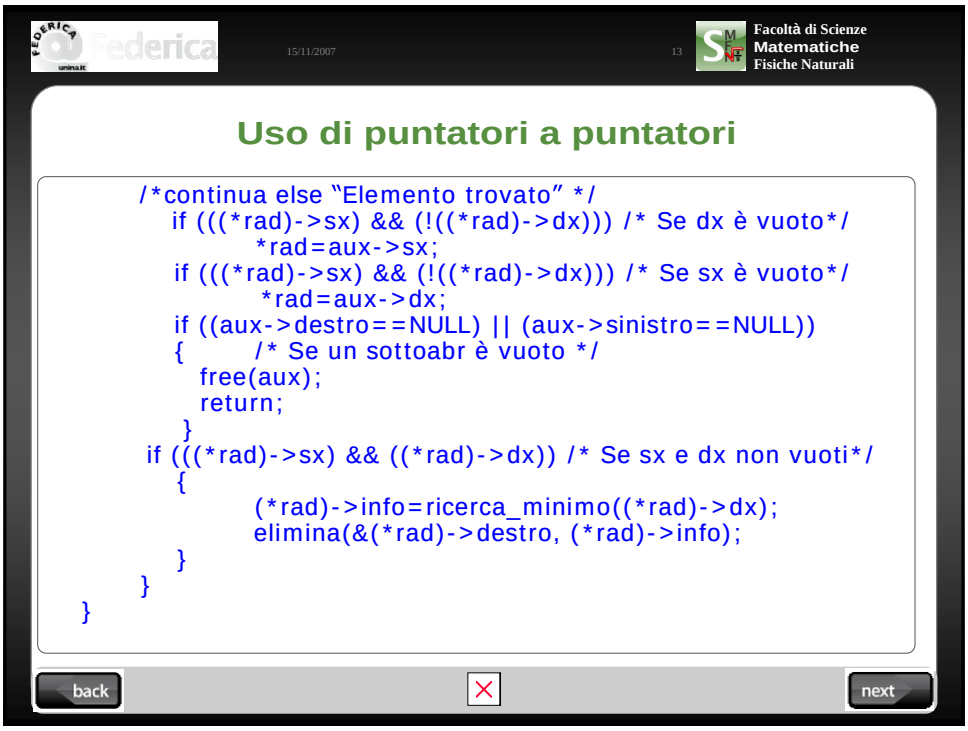
```
void elimina(struct nodo **rad, int el)
{ struct nodo *aux;
  aux=*rad;
  if (!(*rad)) {
    if ((*rad)->info>el)
      /* cerca el a sx */
      elimina(&((*rad)->sx), el);
    else if ((*rad)->info<el)
            /* cerca el a dx */
            elimina(&((*rad)->dx),el);
         else { /* Elemento trovato */
              if (!((*rad)->sx) && (!((*rad)->dx)))
                { /* el e' una foglia */
                  free(*rad);
                  *rad=NULL;
                 }
```

**rad

10	4000	5000
(*rad)->info	(*rad)->sx	(*rad)->dx
3000	3002	3004
	&((*rad)->sx)	&((*rad)->dx)

In riferimento all'esempio di struttura precedente

back next


Federica
15/11/2007
13
Facoltà di Scienze
Matematiche
Fisiche Naturali
Uso di puntatori a puntatori
/*continua else "Elemento trovato" */
if (((*rad)->sx) && (!((*rad)->dx))) /* Se dx è vuoto*/
*rad=aux->sx;
if (((*rad)->sx) && (!((*rad)->dx))) /* Se sx è vuoto*/
*rad=aux->dx;
if ((aux->destro==NULL) || (aux->sinistro==NULL))
{ /* Se un sottoabr è vuoto */
free(aux);
return;
}
if (((*rad)->sx) && ((*rad)->dx)) /* Se sx e dx non vuoti*/
{
(*rad)->info=ricerca_minimo((*rad)->dx);
elimina(&(*rad)->destro, (*rad)->info);
}
}
}
back
next