Azzolini Riccardo 2019-04-29

Operazioni sugli alberi binari di ricerca

1 Ricerca

Procedura RICERCA(v, x)
  begin
    if $v =$ `NULL` **then return** `NULL`;
    if $x = \text{Key}(v)$ **then return** v;
    if $x < \text{Key}(v)$ **then return** RICERCA($\text{sx}(v), x$)
      else return RICERCA($\text{dx}(v), x$);
  end

Quest'operazione cerca un nodo con chiave x nell'albero avente radice v:

- se x è presente nell'albero, viene restituito il nodo corrispondente;
- altrimenti, la procedura restituisce `NULL`.

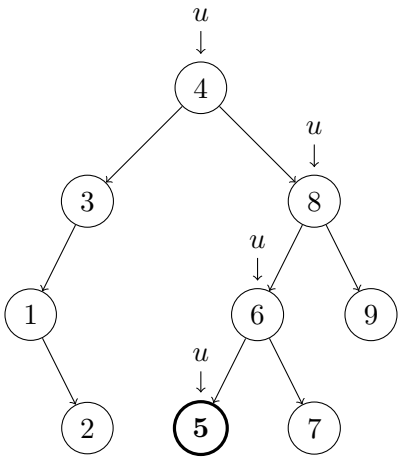
1.1 Versione iterativa

La procedura RICERCA è implementata mediante la ricorsione in coda. Di conseguenza, la si può riscrivere in versione iterativa senza usare strutture dati aggiuntive:

Procedura RICERCAITERATIVA(v, x)
  begin
    $u \coloneqq v$;
    while $u \neq$ `NULL` **do**
      begin
        if $x = \text{Key}(u)$ **then return** u;
        if $x < \text{Key}(u)$ **then** $u \coloneqq \text{sx}(u)$
          else $u \coloneqq \text{dx}(u)$;
      end;
    return `NULL`;
  end

1.1.1 Esempio

Ricerca iterativa di 5:



1.2 Complessità

Il tempo di calcolo, sia per la versione ricorsiva che per quella iterativa, è $O(n)$:

- $O(1)$ nel caso migliore, cioè quando l'elemento cercato si trova alla radice;
- $\Theta(h)$, cioè costo pari all'altezza h dell'albero, nel caso peggiore: se l'albero è degenere, allora l'altezza, e quindi il costo, è $\Theta(n)$.

Osservazione: Una ricerca senza successo si ferma sempre in un nodo che ha al massimo un figlio, perché in un nodo con due figli si può sempre continuare a cercare, sia a sinistra che a destra. Invece, una ricerca con successo si può fermare in qualsiasi nodo dell'albero.

2 Ricerca del minimo

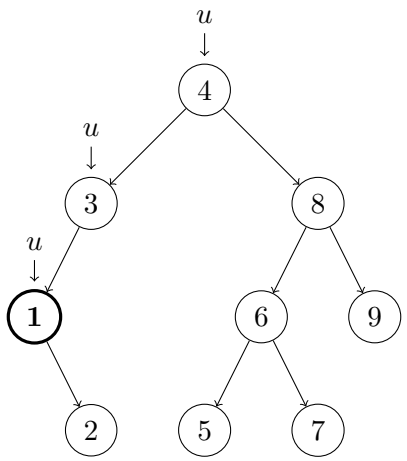
```
Procedura MIN(v)
    begin
        if v = NULL then return NULL;
        u := v;
        while sx(u) ≠ NULL do
            u := sx(u);
        return u;
    end
```

Se l'albero è vuoto, MIN restituisce `NULL`. Altrimenti, si scende a sinistra il più possibile, fino a trovare un nodo senza figlio sinistro, il quale viene restituito.

Questa procedura è già implementata nella versione iterativa.

La ricerca del massimo si effettua in modo analogo, scendendo a destra invece che a sinistra.

2.1 Esempio



2.2 Complessità

Il tempo di calcolo è $O(n)$:

- $O(1)$ nel caso migliore, che si verifica quando la radice contiene il minimo, cioè non ha un sottoalbero sinistro (e anche quando l'albero è vuoto);
- $\Theta(n)$ nel caso peggiore, corrispondente a un albero degenere con solo figli sinistri (una sorta di lista concatenata "pendente" a sinistra).

3 Stampa ordinata

Per stampare in ordine i valori contenuti in un BST è sufficiente visitarlo in ordine simmetrico (in-order):

```
Procedura StampaOrd(v)
    if v ≠ NULL then
        begin
            StampaOrd(sx(v));
            print Key(v);
            StampaOrd(dx(v));
        end
```

Il tempo di calcolo è $\Theta(n)$ in ogni caso.

4 Inserimento

```
Procedura INSERT(v, x)
    begin
        if x < Key(v) then
            if sx(v) ≠ NULL then INSERT(sx(v), x)
                            else CREANODOSX(v, x);
        if x > Key(v) then
            if dx(v) ≠ NULL then INSERT(dx(v), x)
                            else CREANODODX(v, x);
    end

Procedura CREANODOSX(v, x)
    begin
        u := CREANODO(x);
        sx(v) := u;
        padre(u) := v;
        sx(u) := NULL;
        dx(u) := NULL;
    end
Procedura CREANODODX(v, x)
    begin
        u := CREANODO(x);
        dx(v) := u;
        padre(u) := v;
        sx(u) := NULL;
        dx(u) := NULL;
    end
```

Come molte altre operazioni sui BST, l'inserimento è composto da due fasi:

1. la discesa fino al punto in cui inserire il nuovo nodo;
2. la creazione e l'inserimento vero e proprio del nodo nella posizione individuata alla fase 1.

Osservazioni:

- Il nodo inserito è sempre una foglia, e viene aggiunto come figlio sinistro/destro a un nodo che prima non aveva tale figlio.
- In quest'implementazione, ogni nodo è dotato di un riferimento al padre, padre(u): esso non è indispensabile, ma facilita alcune operazioni, a scapito dello spazio occupato da un riferimento in più per ogni nodo.

- Quando l'albero contiene già un nodo con la chiave specificata, non viene effettuato alcun inserimento. Se, invece, si volesse consentire l'inserimento di dati con chiavi duplicate, si potrebbe scegliere tra:
  - cambiare leggermente la definizione di albero binario di ricerca, stabilendo che i nodi nel sottoalbero sinistro di v hanno chiavi $\leq \text{Key}(v)$ (o, equivalentemente, che i nodi nel sottoalbero destro hanno chiavi $\geq \text{Key}(v)$), per determinare dove vengono posizionati i nodi con chiavi uguali;
  - memorizzare in ogni nodo un riferimento a una struttura dati contenente tutti i dati con la stessa chiave (ad esempio, una lista concatenata, oppure un altro BST che utilizza un attributo diverso dei dati come chiave).

4.1 Versione iterativa

L'implementazione iterativa dell'inserimento si ottiene mediante l'eliminazione della ricorsione in coda:

```
Procedura InsertIterativo(v, x)
    if v = NULL then
        begin
            v := CreaNodo(x);
            padre(v) := NULL;
            sx(v) := NULL;
            dx(v) := NULL;
        end
    else
        begin
            while v ≠ NULL do
                begin
                    w := v;
                    if x < Key(v) then v := sx(v)
                                  else v := dx(v);
                end;
            if x < Key(w) then CreaNodoSx(w, x)
                          else CreaNodoDx(w, x);
        end
```

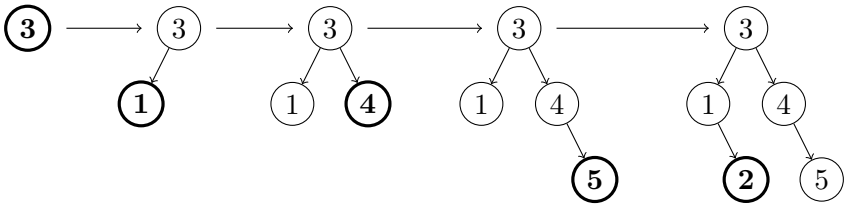
Questa versione utilizza due riferimenti per la discesa nell'albero.

- v server per cercare la posizione in cui effettuare l'inserimento: viene aggiornato, facendolo scendere ogni volta a sinistra o a destra, fino a quando diventa `NULL`, segnalando che la ricerca è terminata.

- w punta di volta in volta al padre di v: al termine della ricerca, si ferma sull'ultimo nodo incontrato durante la discesa, che è quello contenente il riferimento da aggiornare per l'inserimento.

4.2 Esempio

Si vuole costruire un BST, mediante inserimenti successivi, a partire dalla sequenza $3, 1, 4, 5, 2$.



5 Cancellazione

Per la cancellazione, si opera in modo diverso a seconda di quanti figli ha il nodo da eliminare:

- se è una foglia, basta annullare il riferimento a tale nodo nel padre;
- se ha un solo figlio, si sostituisce il nodo con il suo unico sottoalbero;
- se ha due figli, si sostituisce il dato contenuto nel nodo con il valore maggiore del sottoalbero sinistro (o, equivalentemente, con il valore minore del sottoalbero destro), e poi si elimina il nodo da cui è stato ottenuto tale valore, che ha al massimo un figlio (essendo il massimo/minimo di un sottoalbero, non ha figlio destro/sinistro).

```
Procedura REPLACE(v, u)
    begin
        if sx(padre(v)) = v then sx(padre(v)) := u
                           else dx(padre(v)) := u;
        if u ≠ NULL then padre(u) := padre(v);
    end

Procedura REMOVE(v)        // v ha al massimo un figlio
    if FOGLIA(v) then
        REPLACE(v, NULL);
    else
        begin
            if sx(v) ≠ NULL then u := sx(v)
```

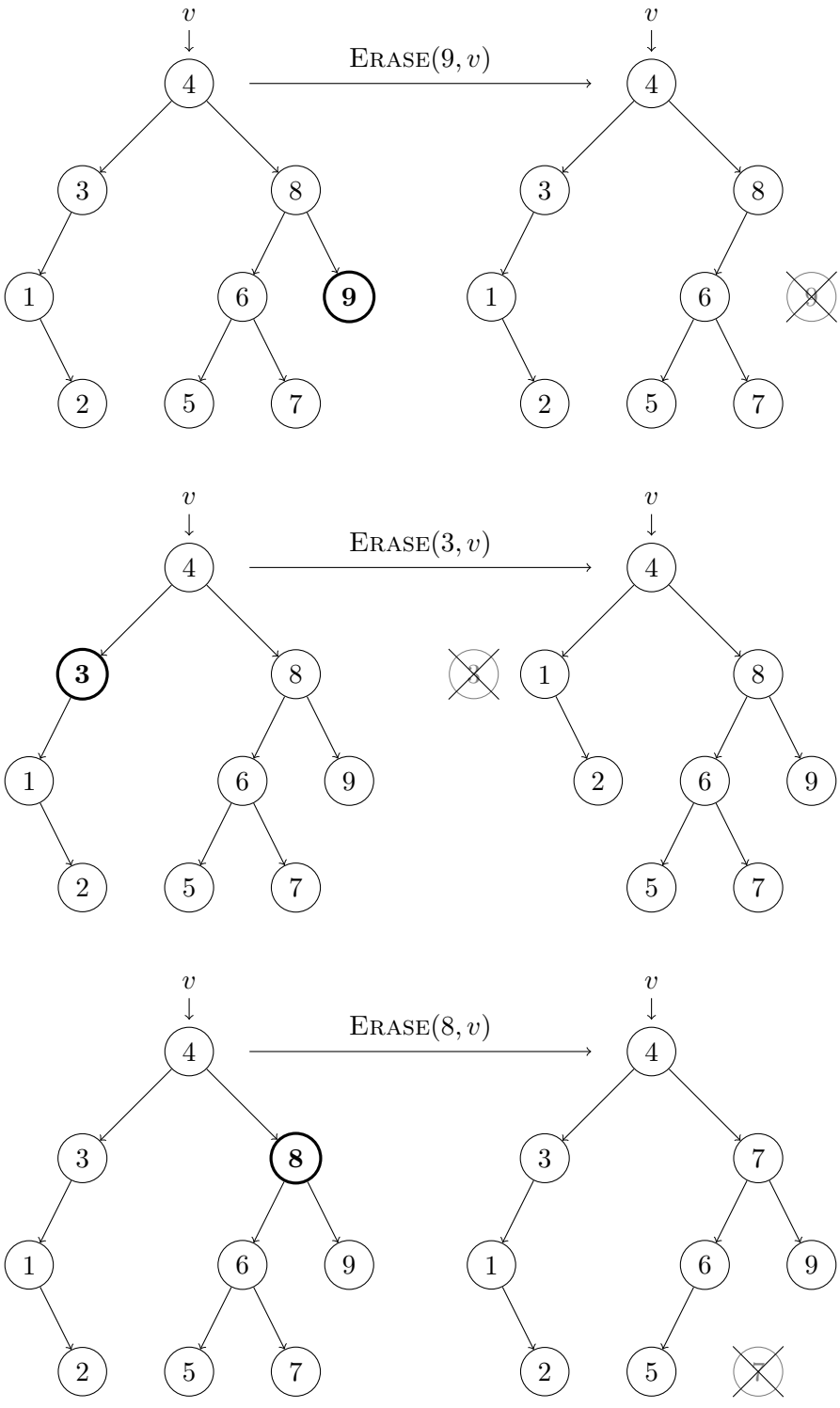
```
                    else u := dx(v);
        if padre(v) = NULL then
            begin      // u diventa la radice
                padre(u) := NULL;
                v := u;
            end
        else
            Replace(v, u);
    end

Procedura Erase(x, v)
    if v ≠ NULL then
        begin
            if x < Key(v) then Erase(x, sx(v));
            if x > Key(v) then Erase(x, dx(v));
            if x = Key(v) then
                if Figli(v) < 2 then
                    Remove(v)
                else
                    begin
                        v_M := Max(sx(v));
                        Key(v) := Key(v_M);
                        Remove(v_M);
                    end;
        end
```

- La procedura Replace mette il nodo u al posto di v (come figlio di padre(v)).
- Remove elimina un nodo con al massimo un figlio, usando Replace per sostituirlo con l'eventuale figlio, oppure con un riferimento nullo.

  Come caso particolare, se il nodo da eliminare è la radice, non si può utilizzare la procedura Replace (dato che essa opera sul padre del nodo da sostituire, e la radice non ha un padre), quindi si aggiorna direttamente la variabile v (che deve essere passata per riferimento) per cambiare la radice dell'albero.
- Erase effettua innanzitutto una ricerca (mediante ricorsione in coda) per individuare il nodo da cancellare.
  - Se esso non viene trovato, dopo aver raggiunto una foglia si effettua la chiamata Erase(x, NULL), nella quale viene saltato il corpo dell'if, e quindi la ricerca termina.
  - Se, invece, si trova un nodo contenente il dato da eliminare, esso viene cancellato. In particolare, quando tale nodo ha due figli, come valore sostitutivo si sceglie il massimo del sottoalbero sinistro.

5.1 Esempi


v
ERASE(9, v)
4
3
8
1
6
9
2
5
7
v
4
3
8
1
6
8
2
5
7
v
ERASE(3, v)
4
3
8
1
6
9
2
5
7
v
4
3
8
1
6
9
2
5
7
v
ERASE(8, v)
4
3
8
1
6
9
2
5
7
v
4
3
7
1
6
9
2
5
7

5.2 Complessità

Le procedure REPLACE e REMOVE richiedono tempo di calcolo $O(1)$.

Il costo di ERASE, invece, è $O(n)$:

- nel caso migliore, quando il nodo cancellato è la radice e ha al massimo un figlio, il costo è $O(1)$;
- se il nodo cancellato ha due figli, il costo dell'operazione corrisponde alla lunghezza totale dei percorsi dalla radice a tale nodo e da esso al suo sostituto: questa lunghezza è sempre minore o uguale all'altezza h dell'albero, quindi il costo nel caso peggiore è $\Theta(h)$, e in particolare $\Theta(n)$ se l'albero è degenere.