Titolo unità didattica: Strutture dati dinamiche gerarchiche [P2_09]

Titolo: alberi binari di ricerca (search binary trees) [5-T]

Definizioni ed algoritmi di gestione

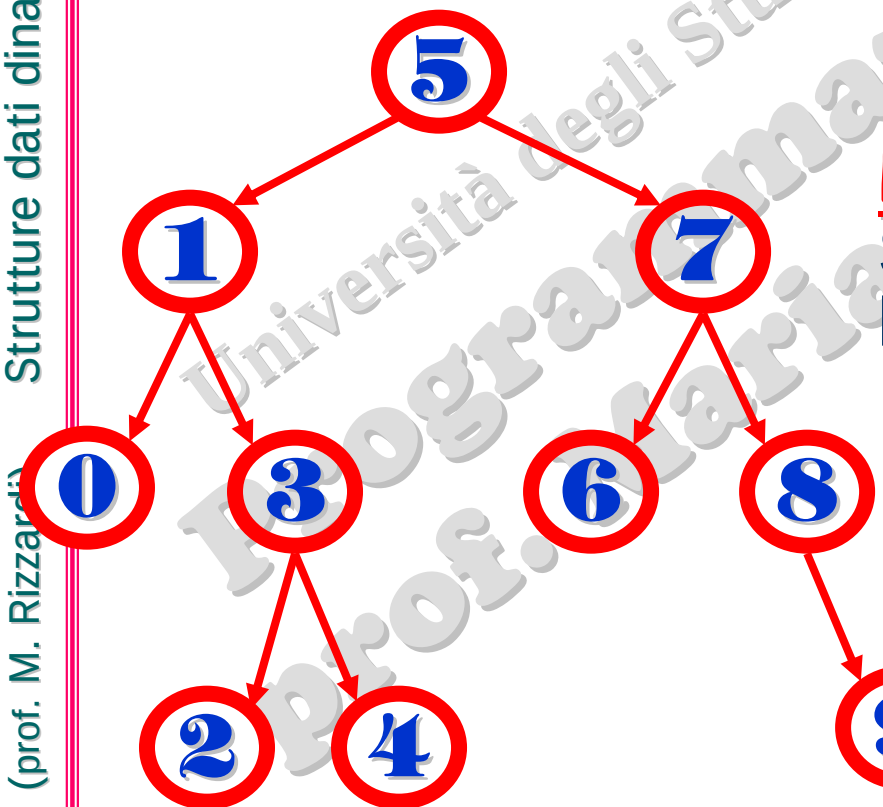
Argomenti trattati:

- ✓ Definizione e proprietà di un albero binario di ricerca
- ✓ Algoritmo per la costruzione e visita di un albero binario di ricerca
- ✓ Algoritmo ricorsivo di ricerca binaria su un albero binario di ricerca

Prerequisiti richiesti: alberi binari

Alberi binari ordinati secondo una chiave (o Alberi Binari di Ricerca – Search Binary Trees - SBT)

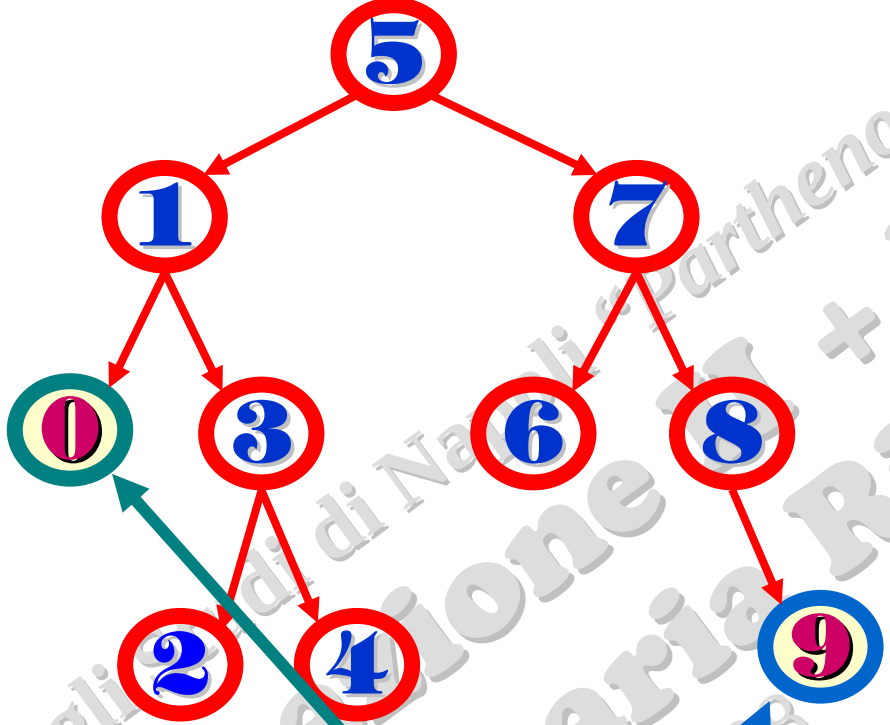
- È un particolare albero binario in cui la visita inorder (in ordine simmetrico) consente di accedere alle informazioni in ordine crescente di chiave (key(nodo));
- È un particolare albero binario dove la ricerca binaria risulta particolarmente semplice;



Proprietà di un SBT:

Se y è un qualsiasi nodo dell'albero si ha

- ∀ nodo x del sottoalbero sinistro di y
  $key(x) \leq key(y)$
- ∀ nodo z del sottoalbero destro di y
  $key(y) \leq key(z)$

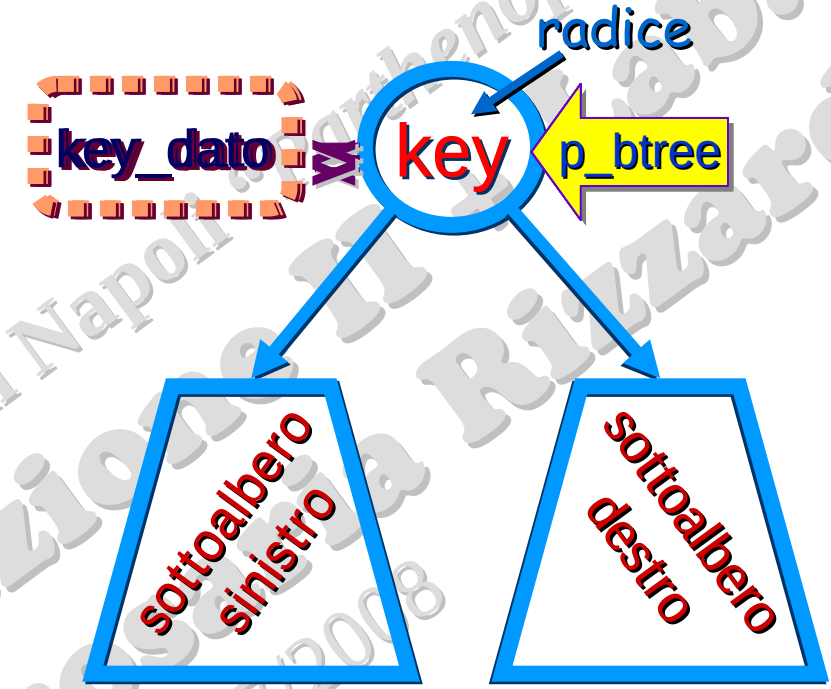


In un albero binario di ricerca con chiavi tutte diverse, si ha

- la chiave di **valore minimo** occupa il nodo foglia del sottoalbero più a sinistra (cioè la prima foglia che si incontra con la visita inorder);
- la chiave di **valore massimo** occupa il nodo foglia del sottoalbero più a destra (cioè l'ultima foglia che si incontra con la visita inorder);

Algoritmo ricorsivo di ricerca binaria

come tutti gli algoritmi della classe "divide et impera" anche la ricerca binaria si esprime molto semplicemente in notazione ricorsiva




```
char function b_search_ric(KeyType key_dato, b_tree ** p_btree)
if (*p_btree) == NULL                          return 0    /* cioè falso – non trovato! */
else
      if key_dato == (*p_btree)->key           return 1    /* cioè vero – trovato! */
      else
            if key_dato < (*p_btree)->key                          /* continua la ricerca */
                *p_btree = (*p_btree)->pt_sx;
                return b_search_ric(key_dato, p_btree)      /* nel sottoalbero sinistro */
            else
                *p_btree = (*p_btree)->pt_dx;
                return b_search_ric(key_dato, p_btree)      /* nel sottoalbero destro */
```

La costruzione di un albero binario di ricerca avviene con un algoritmo tipo ricerca binaria

- inserisci nodo radice;
- while esistono nodi da inserire
  - input di un nodo N;
  - nodo_corr := radice;
  - while key(N) ≤ key(nodo_corr) & c'è sottoalbero sinx
    - nodo_corr := figlio_sinx(nodo_corr);

    endwhile
  - while key(N) ≥ key(nodo_corr) & c'è sottoalbero dx
    - nodo_corr := figlio_dx(nodo_corr);

    endwhile
  - if key(N) ≤ key(nodo_corr)

      inserisci nodo N come figlio_sinx di nodo_corr;

    else

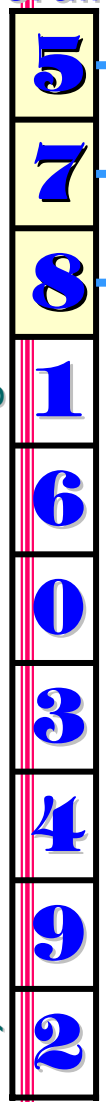
      inserisci nodo N come figlio_dx di nodo_corr;

    endif

  endwhile

Esempio

ordine di arrivo



1°

2°

3°

- inserisci nodo radice;
- while esistono nodi da inserire
  - input di un nodo N;
  - nodo_corr := radice;
  - while key(N) ≤ key(nodo_corr) & c'è sottoalbero sinx
    - nodo_corr := figlio_sinx(nodo_corr);

    endwhile
  - while key(N) ≥ key(nodo_corr) & c'è sottoalbero dx
    - nodo_corr := figlio_dx(nodo_corr);

    endwhile
  - if key(N) ≤ key(nodo_corr)

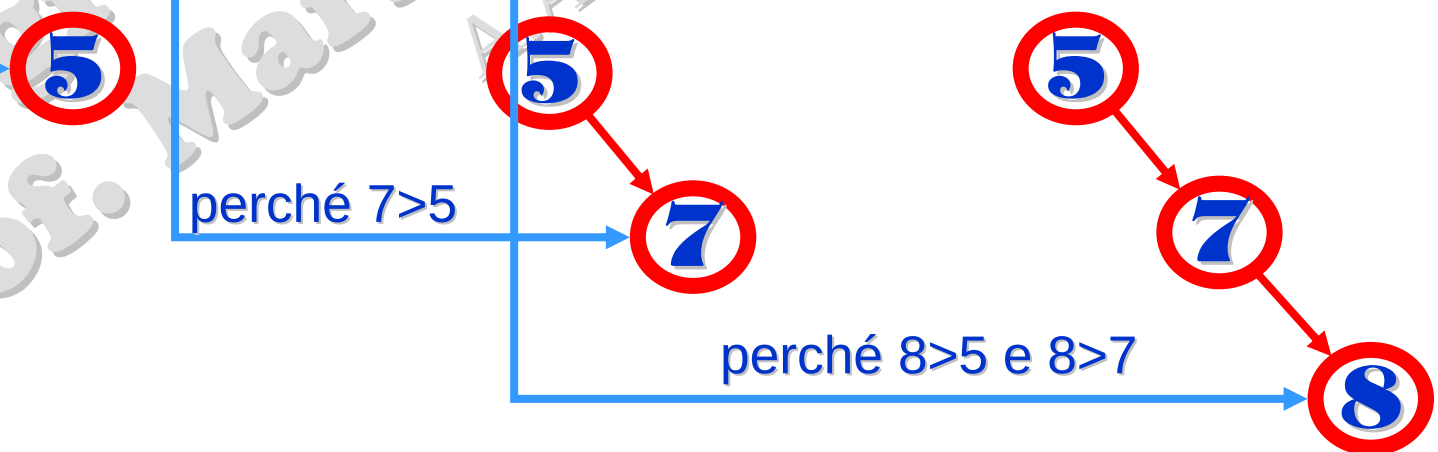
      inserisci nodo N come figlio_sinx di nodo_corr;

    else

      inserisci nodo N come figlio_dx di nodo_corr;

    endif

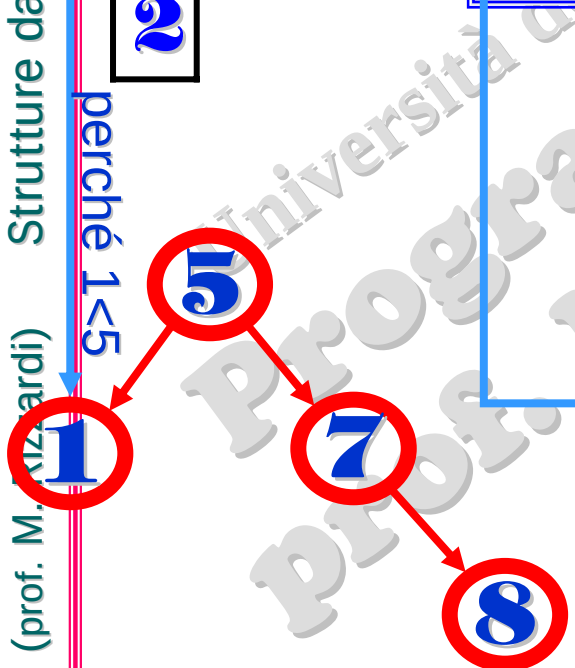
  endwhile

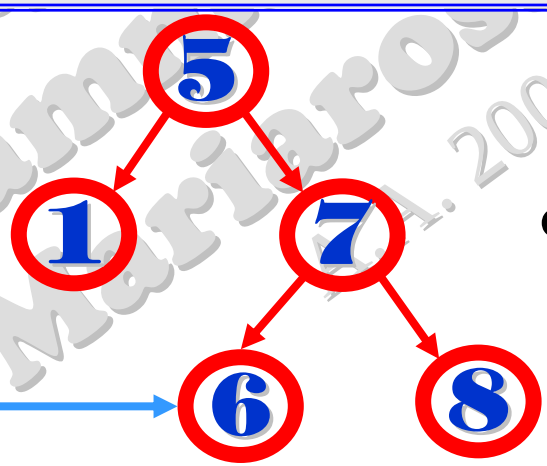


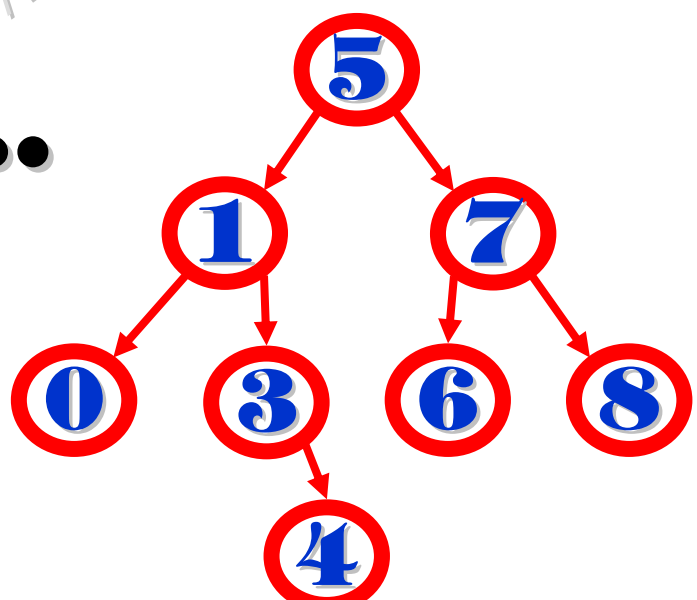
ordine di arrivo


```
while esistono nodi da inserire
    input di un nodo N;
    nodo_corr := radice;
    while key(N) ≤ key(nodo_corr) & c'è sottoalbero sinx
            nodo_corr := figlio_sinx(nodo_corr);
    endwhile
    while key(N) ≥ key(nodo_corr) & c'è sottoalbero dx
            nodo_corr := figlio_dx(nodo_corr);
    endwhile
    if key(N) ≤ key(nodo_corr)
        inserisci nodo N come figlio_sinx di nodo_corr;
    else
        inserisci nodo N come figlio_dx di nodo_corr;
    endif
endwhile
```

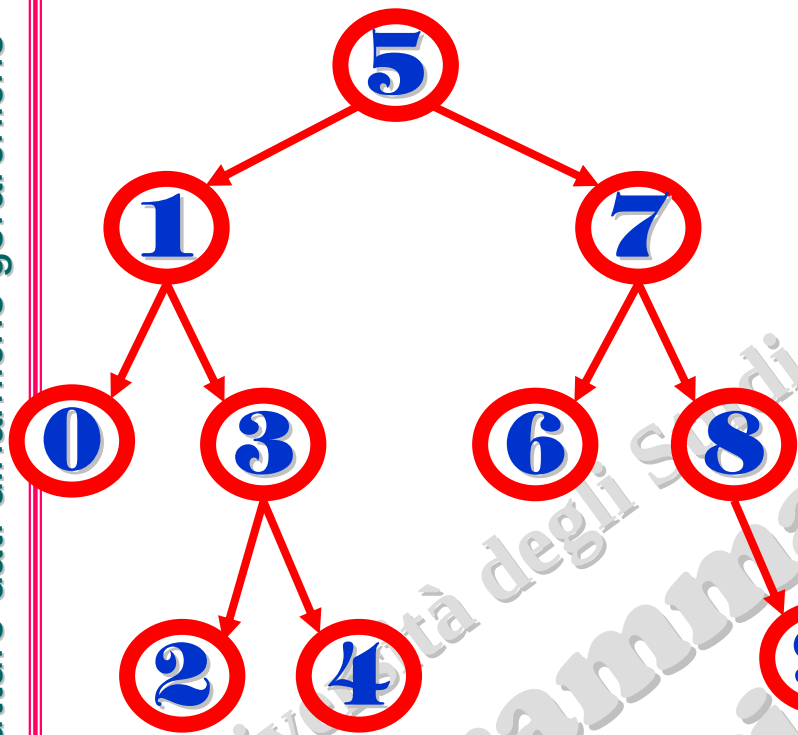
perché 1<5







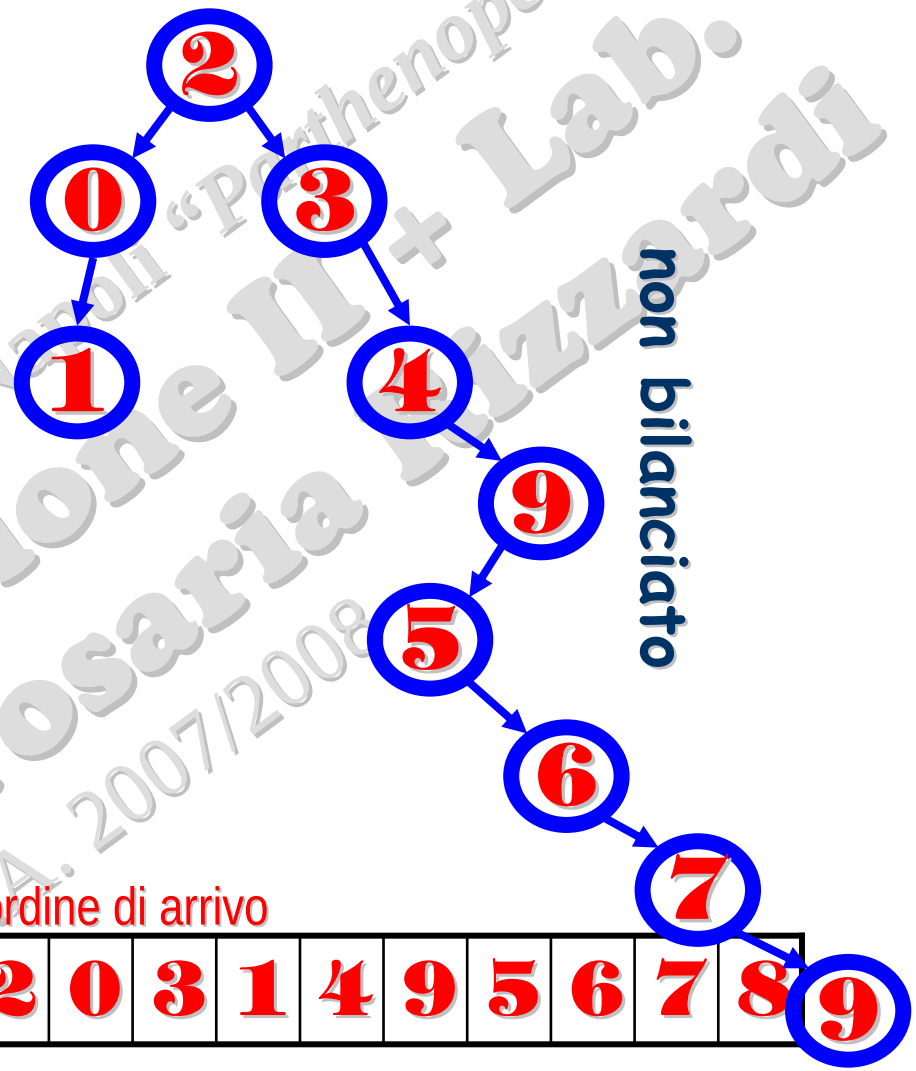
Il bilanciamento dell'albero binario di ricerca dipende dall'ordine (casuale) di arrivo dei nodi (*)



bilanciato

ordine di arrivo

5	7	8	1	6	0	3	4	9	2



non bilanciato

ordine di arrivo

2	0	3	1	4	9	5	6	7	8

(*) L'operazione di rotazione di nodi negli alberi Red-Black consente il bilanciamento di un albero binario

Esercizi

1 Scrivere *function* C iterativa per la costruzione di un albero binario di ricerca implementato mediante array.

[liv. 2]

2 Scrivere *function C* iterativa per la costruzione di un albero binario di ricerca implementato mediante liste multiple. [liv. 3]