POLITECNICO
DI TORINO
Dipartimento
di Automatica e Informatica

Gli Interval BST

Paolo Camurati e Gianpiero Cabodi

Interval BST

Intervallo chiuso: coppia ordinata di reali $[t_1, t_2]$, dove $t_1 \leq t_2$ e $[t_1, t_2] = \{t \in \Re : t_1 \leq t \leq t_2\}$.

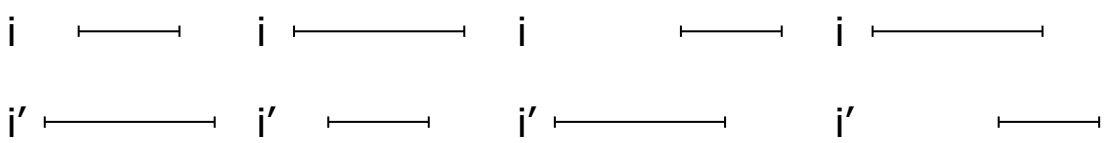
L'item intervallo $[t_1, t_2]$ può essere realizzato da una `struct` con campi `low` = t_1 e `high` = t_2. Gli intervalli i e i' hanno intersezione se e solo se:

`low[i]` $\leq$ `high[i']` && `low[i']` $\leq$ `high[i]`.

$\forall$ i, i' vale la seguente tricotomia:

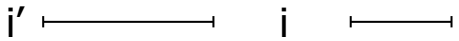
a. i e i' hanno intersezione
b. `high[i]` $\leq$ `low[i']`
c. `high[i']` $\leq$ `low[i]`

caso a

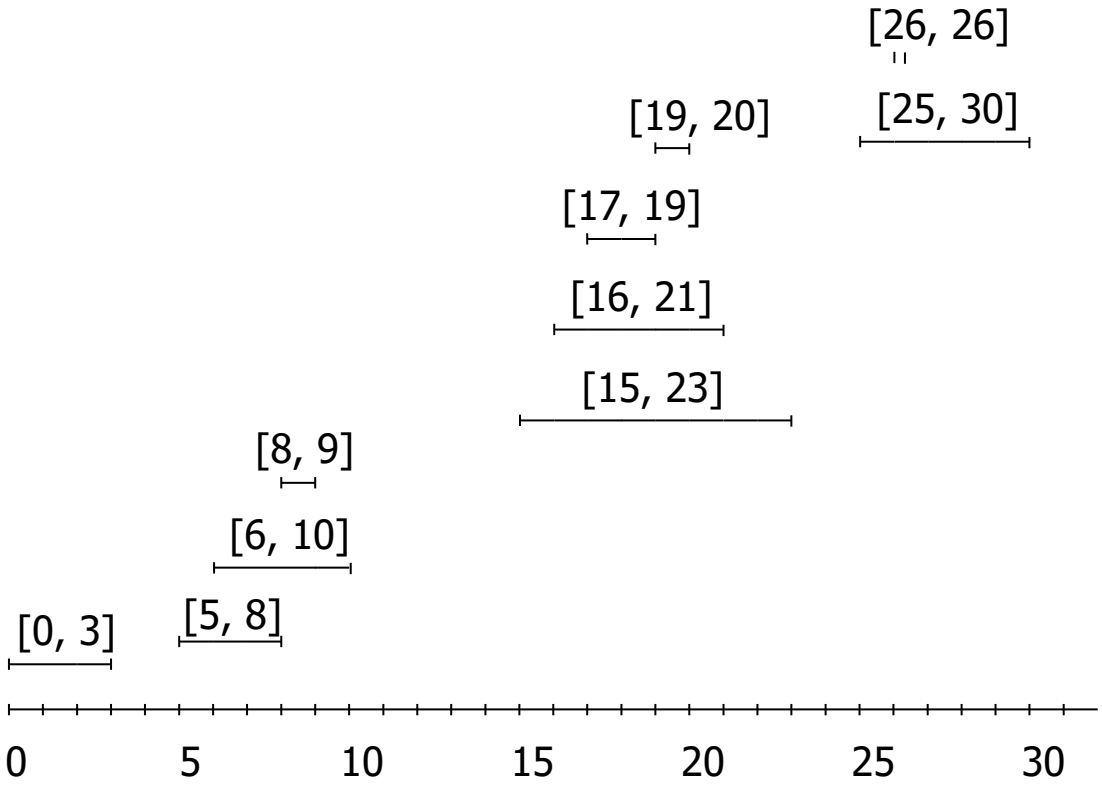


caso b

caso c



Esempio



Procedura

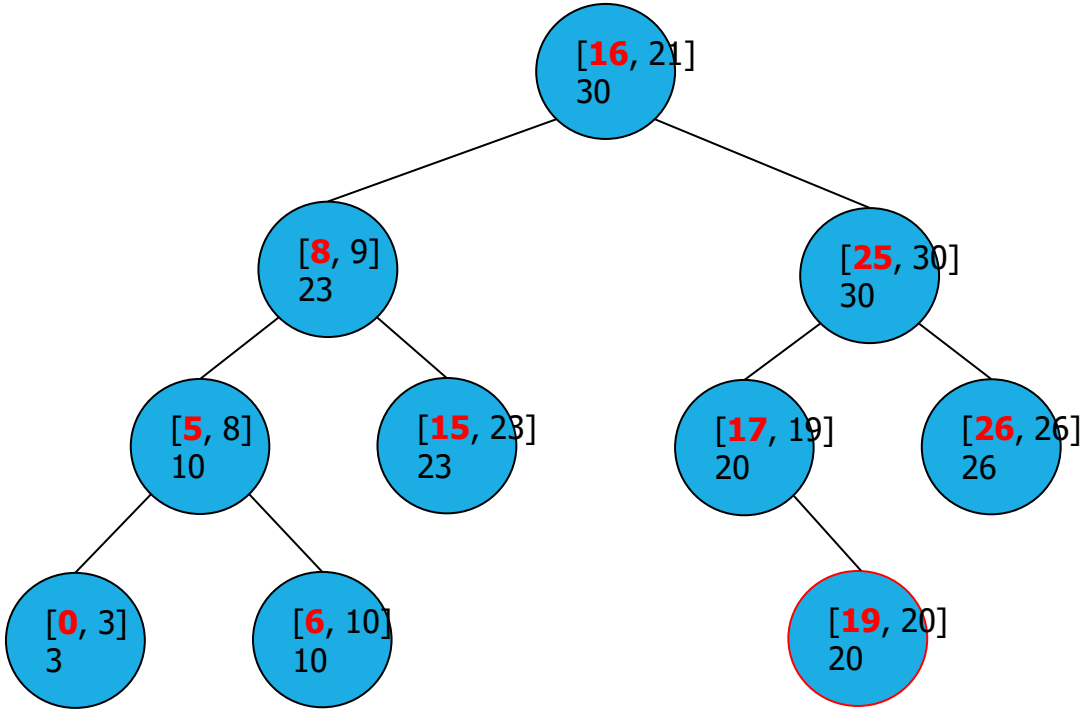
1. identificare la struttura dati candidata
2. identificare le informazioni supplementari
3. verificare di poter mantenere le informazioni supplementari senza alterare la complessità delle operazioni esistenti
4. sviluppare nuove operazioni.

BST con inserzione secondo l'estremo inferiore

`max`: massimo `high` del sottoalbero

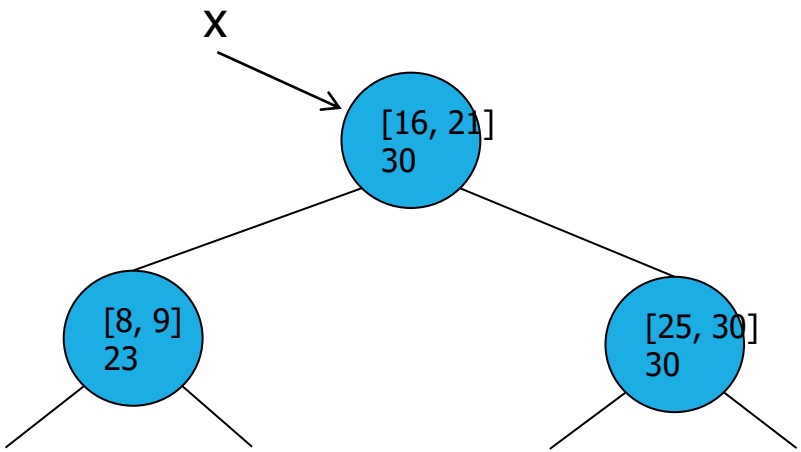
O(1)

```
Item IBSTsearch(BST, Item);
```

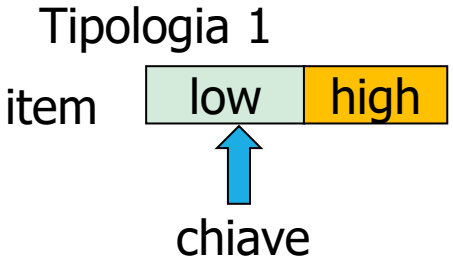

[16, 21]
30
[8, 9]
23
[25, 30]
30
[5, 8]
10
[15, 23]
23
[17, 19]
20
[26, 26]
26
[0, 3]
3
[6, 10]
10
[19, 20]
20

Il calcolo di max è di complessità $\Theta(1)$:

x->max = max (high(x), x->left->max, x->right->max)



Quasi ADT Item



Item.h

```
typedef struct item { int low; int high; } Item;

Item    ITEMscan();
Item    ITEMsetVoid();
int     ITEMcheckVoid(Item val);
void    ITEMstore(Item val);
int     ITEMhigh();
int     ITEMlow();
int     ITEMoverlap(Item val1, Item val2);
int     ITEMeq(Item val1, Item val2);
int     ITEMlt(Item val1, Item val2);
int     ITEMlt_int(Item val1, int val2);
```

estremo superiore

estremo inferiore

intersezione

= in inserzione

< in inserzione

< in ricerca

Item.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "Item.h"

Item ITEMscan() {
  Item val;
  printf("low = "); scanf("%d", &val.low);
  printf("high = "); scanf("%d", &val.high);
  return val;
}

void ITEMstore(Item val) {
  printf("[%d, %d] ", val.low, val.high);
}

Item ITEMsetVoid() {
  Item val = {-1, -1};
  return val;
}
```

```
int ITEMcheckVoid(Item val) {
  if ((val.low == -1) && (val.high == -1))
    return 1;
  return 0;
}

int ITEMhigh(Item val) { return val.high; }

int ITEMlow(Item val) { return val.low; }

int ITEMoverlap(Item val1, Item val2) {
  if ((val1.low <= val2.high) && (val2.low<= val1.high))
    return 1;
  return 0;
}
```

```
int ITEMeq(Item val1, Item val2) {
  if ((val1.low == val2.low) && (val1.high == val2.high))
    return 1;
  return 0;
}

int ITEMlt(Item val1, Item val2) {
  if ((val1.low < val2.low))
    return 1;
  return 0;
}

int ITEMlt_int(Item val1, int val2) {
  if ((val1.low < val2))
    return 1;
  return 0;
}
```

Operazioni

```
Item IBSTsearch(IBST ibst, Item x);
```

cerca un item (intervallo) nell'Interval BST e ritorna il primo intervallo che lo interseca

```
void IBSTinsert(IBST ibst, Item x);
```

inserisci un item (intervallo) nell'Interval BST

```
void IBSTdelete(IBST ibst, Item x);
```

cancella un item (intervallo) dall'Interval BST

ADT di I classe Interval BST

IBST.h

```
typedef struct  intervalbinarysearchtree *IBST;

void     IBSTinit(IBST ibst);
void     IBSTfree(IBST ibst);
void     BSTinsert(IBST ibst, Item x);
void     IBSTdelete(IBST ibst, Item x);
Item     IBSTsearch(IBST ibst, Item x);
int      IBSTcount(IBST ibst);
int      IBSTempty(IBST ibst);
void     IBSTvisit(IBST ibst, int strategy);
```

nuove funzioni

funzioni modificate

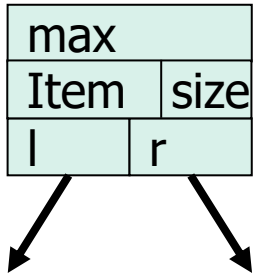
IBST.c

```
#include <stdlib.h>
#include <stdio.h>
#include "Item.h"
#include "IBST.h“

typedef struct IBSTnode *link;

struct IBSTnode {Item item; link l, r; int N; int max;} ;

struct intervalbinarysearchtree {link root;int size;link z;};

static link NEW(Item item, link l, link r, int N, int max) {
  link x = malloc(sizeof *x);
  x->item = item;
  x->l = l; x->r = r;  x->N = N;
  x->max = max;
  return x;
}
```

max: massimo high del sottoalbero

size: dimensione del sottoalbero

IBSTnode



```
static void NODEshow(link x) {
  ITEMstore(x->item); printf("max = %d\n", x->max);
}
IBST IBSTinit( ) {
  IBST ibst = malloc(sizeof *ibst) ;
  ibst->N = 0;
  ibst->root=(ibst->z=NEW(ITEMsetNull(),NULL,NULL,0,-1));
  return ibst;
}
void IBSTfree(IBST ibst) {
  if (ibst == NULL) return;
  treeFree(ibst->root, ibst->z);
  free(ibst->z); free(ibst);
}
static void treeFree(link h, link z) {
  if (h == z) return;
  treeFree(h->l, z); treeFree(h->r, z);
  free(h);
}
```

```
int IBSTcount(IBST ibst) { return ibst->size; }
int IBSTempty(IBST ibst) {
  if (IBSTcount(ibst) == 0) return 1;
  return 0;
}
static void treePrintR(link h, link z, int strategy) {
  if (h == z) return;
  if (strategy == PREORDER)
    NODEshow(h);
  treePrintR(h->l, z, strategy);
  if (strategy == INORDER)
    NODEshow(h);
  treePrintR(h->r, z, strategy);
  if (strategy == POSTORDER)
    NODEshow(h);
}
void IBSTvisit(IBST ibst, int strategy) {
  if (IBSTempty(ibst)) return;
  treePrintR(ibst->root, ibst->z, strategy);
}
```

Insert

```
link insertR(link h, Item item, link z) {
  if (h == z)
    return NEW(item, z, z, 1, ITEMhigh(item));
  if (ITEMlt(item, h->item)) {
    h->l = insertR(h->l, item, z);
    h->max = max(h->max, h->l->max, h->r->max);
  }
  else {
    h->r = insertR(h->r, item, z);
    h->max = max(h->max, h->l->max, h->r->max);
  }
  (h->N)++;
  return h;
}

void IBSTinsert(IBST ibst, Item item) {
  ibst->root = insertR(ibst->root, item, ibst->z);
  ibst->size++;
}
```

rotL/rotR

```
link rotL(link h) {
  link x = h->r;
  h->r = x->l;
  x->l = h;
  x->N = h->N;
  h->N = h->l->N + h->r->N +1;
  h->max = max(ITEMhigh(h->item), h->l->max, h->r->max);
  x->max = max(ITEMhigh(x->item), x->l->max, x->r->max);
  return x;
}
link rotR(link h) {
  link x = h->l;
  h->l = x->r;
  x->r = h;
  x->N = h->N;
  h->N = h->r->N + h->l->N +1;
  h->max = max(ITEMhigh(h->item), h->l->max, h->r->max);
  x->max = max(ITEMhigh(x->item), x->l->max, x->r->max);
  return x;
```

partR/joinLR

```
link partR(link h, int r) {
  int t  = h->l->N;
  if (t > r) {
    h->l = partR(h->l, r);
    h = rotR(h);
  }
  if (t < r) {
    h->r = partR(h->r, r-t-1);
    h = rotL(h);
  }
  return h;
}
link joinLR(link a, link b, link z) {
  if (b == z)  return a;
  b = partR(b, 0);
  b->l = a;
  b->N = a->N + b->r->N +1;
  b->max = max(ITEMhigh(b->item), a->max, b->r->max);
  return b;
}
```

Delete

```
link deleteR(link h, Item item, link z) {
  link x;
  if (h == z) return z;
  if (ITEMlt(item, h->item)) {
    h->l = deleteR(h->l, item, z);
    h->max = max(ITEMhigh(h->item), h->l->max, h->r->max);
  }
  if (ITEMlt(h->item, item)) {
    h->r = deleteR(h->r, item, z);
    h->max = max(ITEMhigh(h->item), h->l->max, h->r->max);
  }
  (h->N)--;
  if (ITEMeq(item, h->item)) {
    x = h; h = joinLR(h->l, h->r, z); free(x);
  }
  return h;
}
void IBSTdelete(IBST ibst, Item item) {
  ibst->root=deleteR(ibst->root,item,ibst->z);
  ibst->size--;
}
```

Search

Ricerca di un nodo `h` con intervallo che interseca l'intervallo `i`:

- percorrimento dell'albero dalla radice
- terminazione: trovato intervallo che interseca `i` oppure si è giunti ad un albero vuoto
- ricorsione: dal nodo `h`
  - su sottoalbero sinistro se

    `h->l->max ≥ low[i]`
  - su sottoalbero destro se

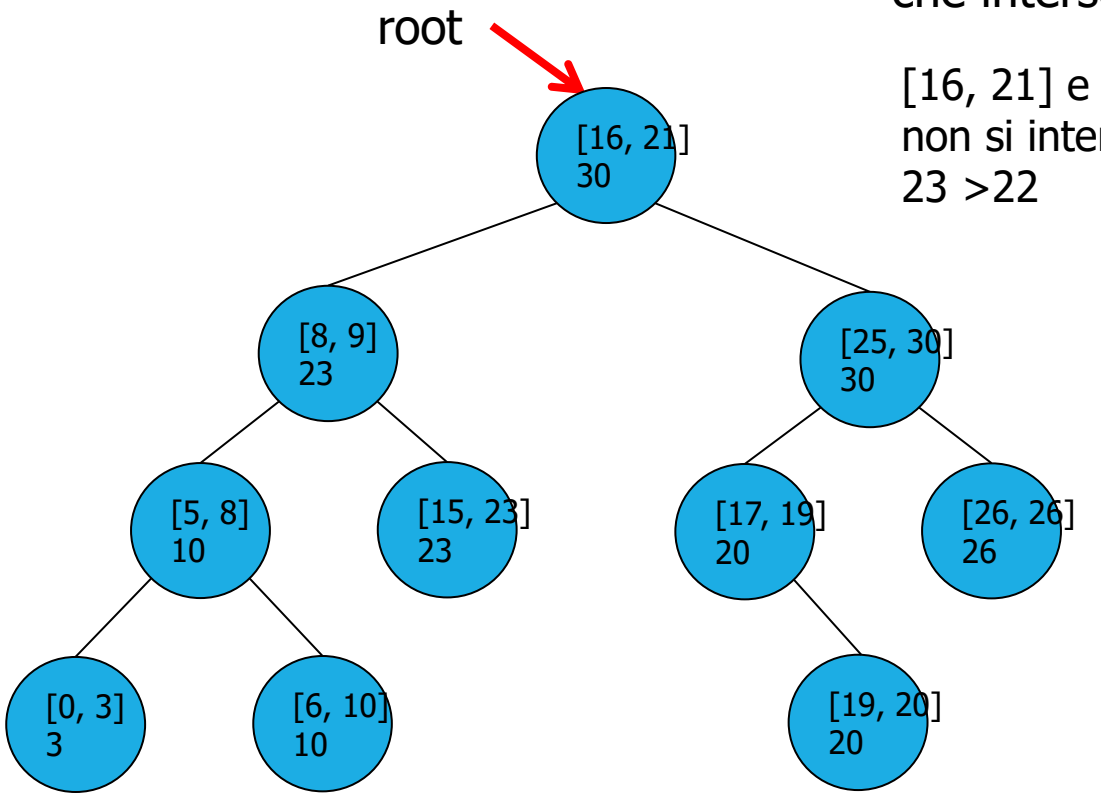
    `h->l->max < low[i]`

```
Item searchR(link h, Item item, link z) {
  if (h == z)
    return ITEMsetNull();
  if (ITEMoverlap(item, h->item))
    return h->item;
  if (ITEMlt_int(item, h->l->max))
    return searchR(h->l, item, z);
  else
    return searchR(h->r, item, z);
  }

Item IBSTsearch(IBST ibst, Item item) {
  return searchR(ibst->root, item, ibst->z);
```

Esempio

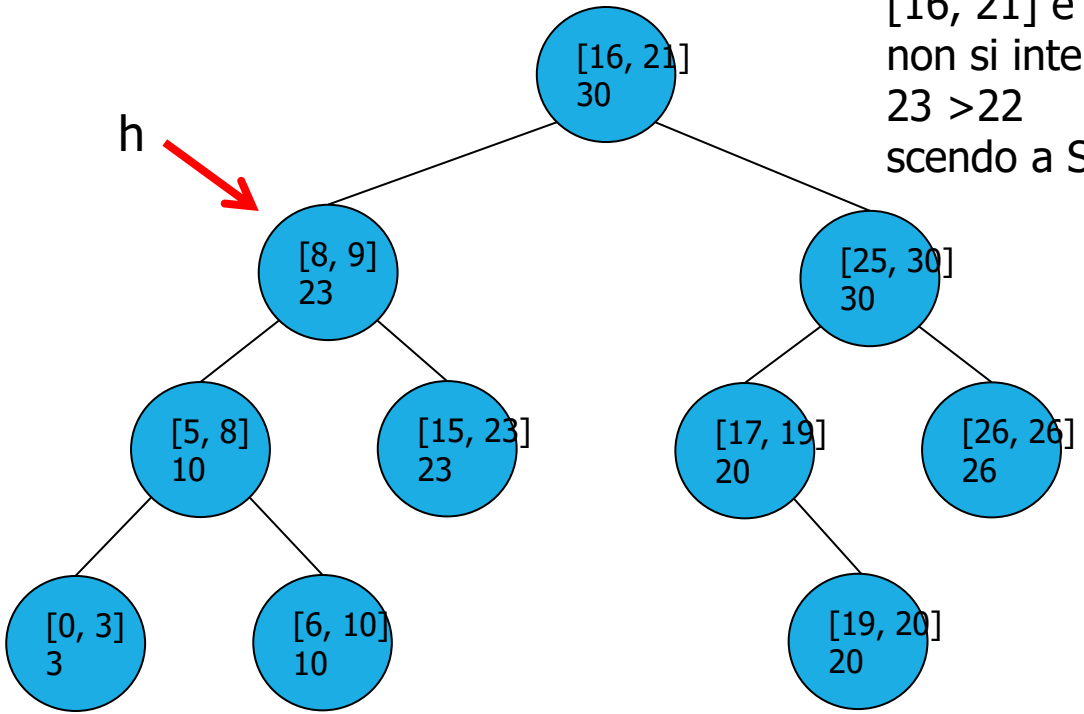
Ricerca di un intervallo che interseca [22, 25]

[16, 21] e [22, 25] non si intersecano
23 >22



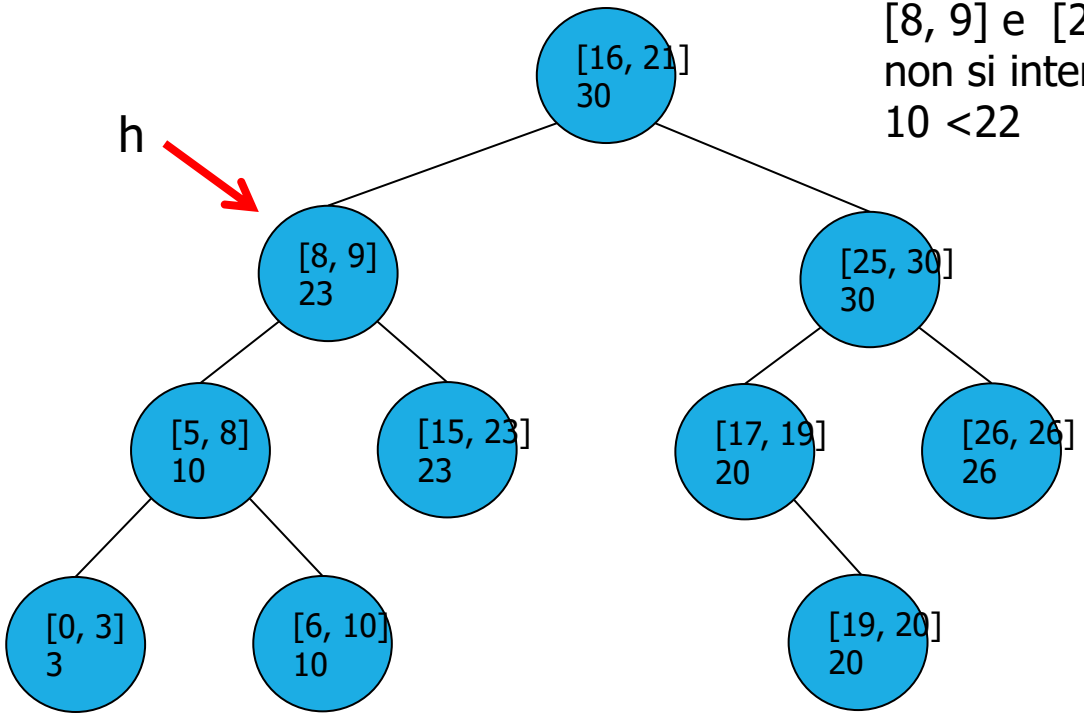
Ricerca di un intervallo che interseca [22, 25]

[16, 21] e [22, 25] non si intersecano
23 >22
scendo a SX



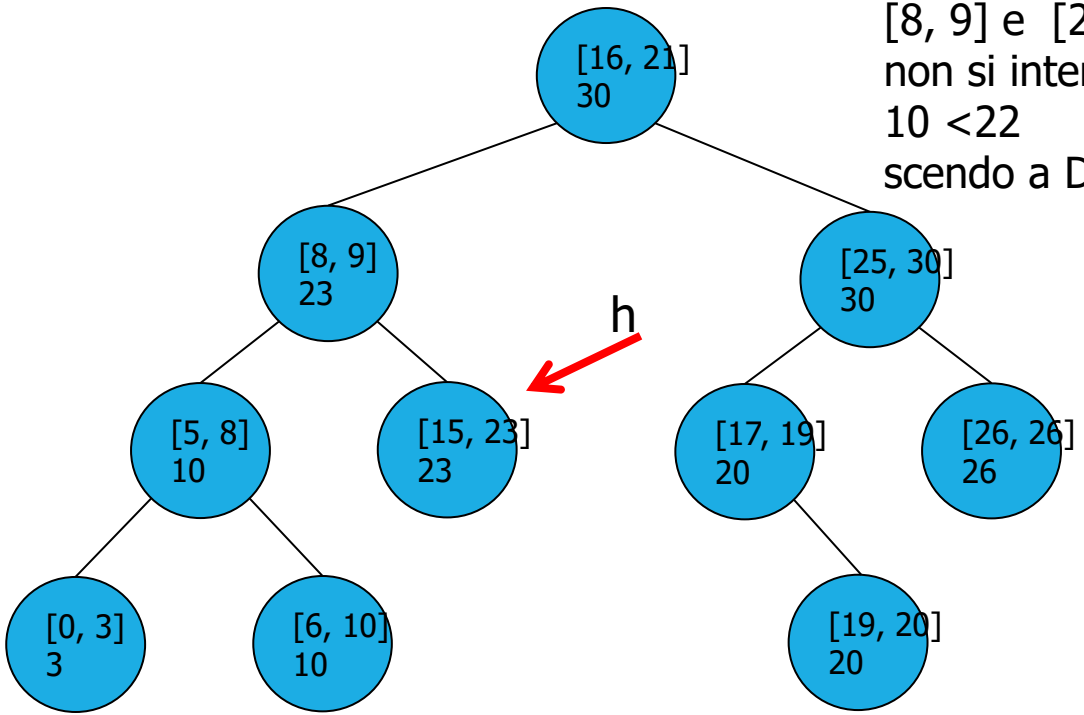
Ricerca di un intervallo che interseca [22, 25]

[8, 9] e [22, 25] non si intersecano
10 <22



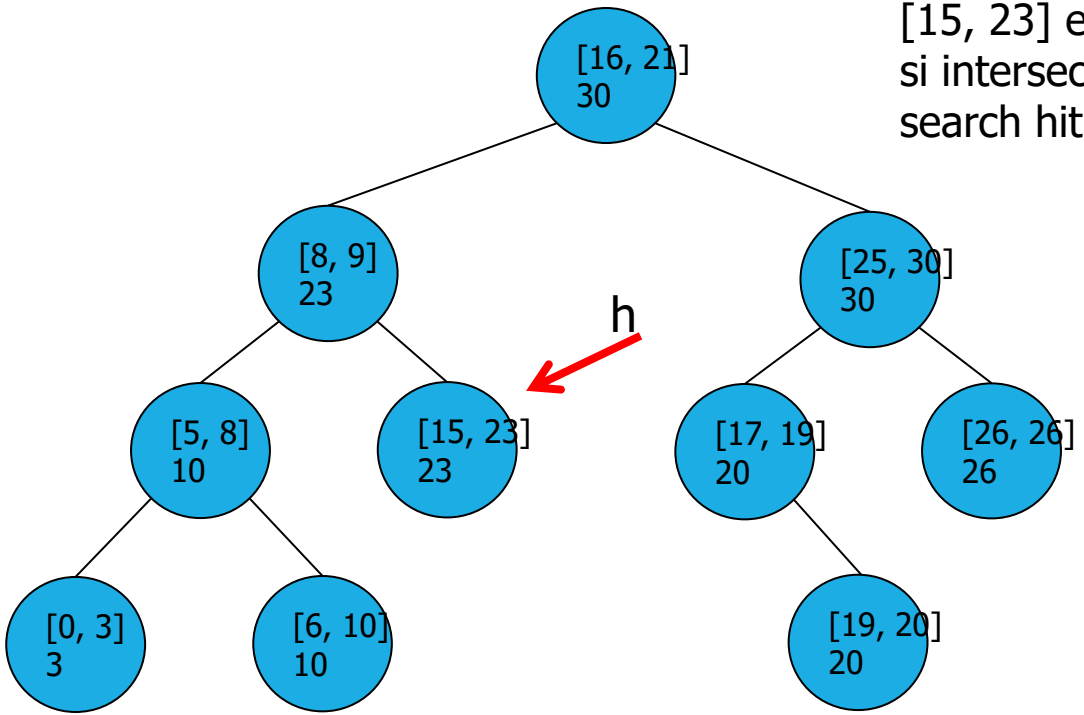
Ricerca di un intervallo che interseca [22, 25]

[8, 9] e [22, 25] non si intersecano
10 <22
scendo a DX



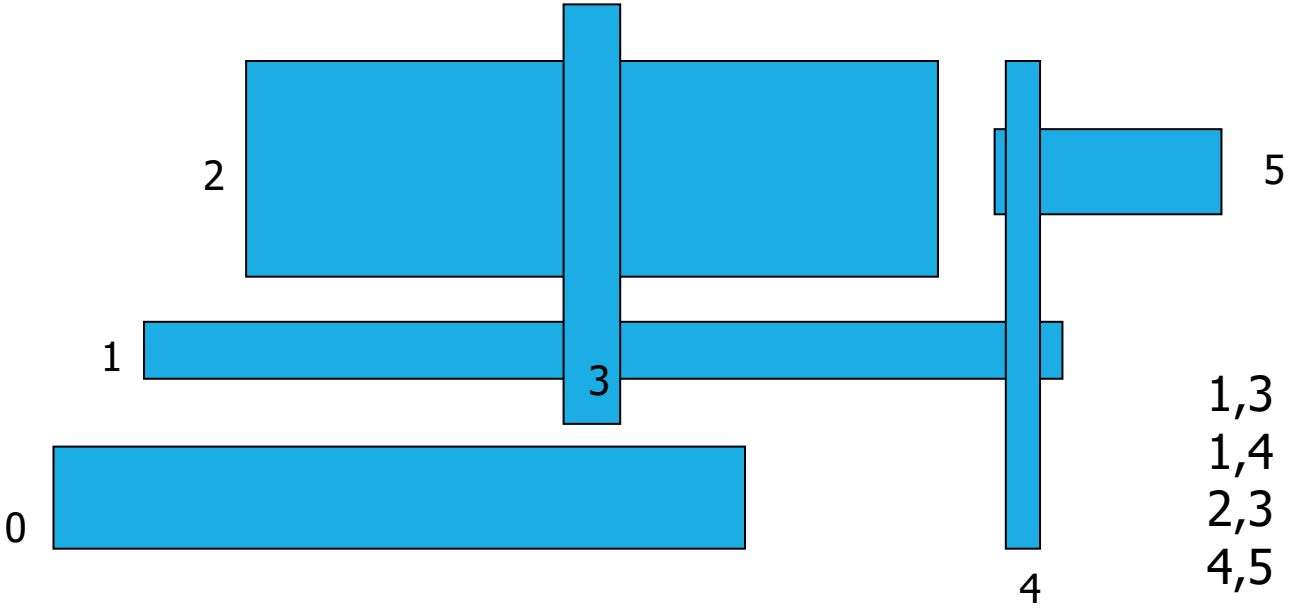
Ricerca di un intervallo che interseca [22, 25]

[15, 23] e [22, 25] si intersecano search hit!



Applicazioni degli I-BST

Dati N rettangoli disposti parallelamente agli assi ortogonali, determinare tutte le coppie che si intersecano:



Applicazione al CAD elettronico

Verificare se le piste si intersecano in un circuito elettronico.

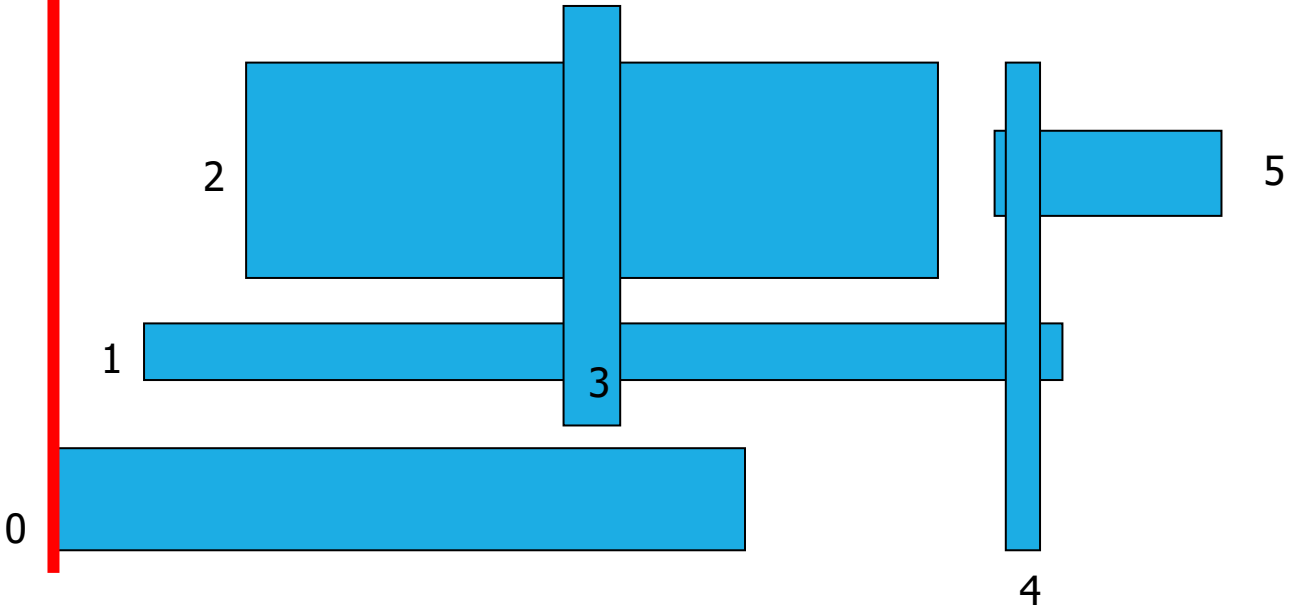
Algoritmo banale: controllare l'intersezione tra tutte le coppie di rettangoli, complessità $O(N^2)$.

Algoritmo efficiente

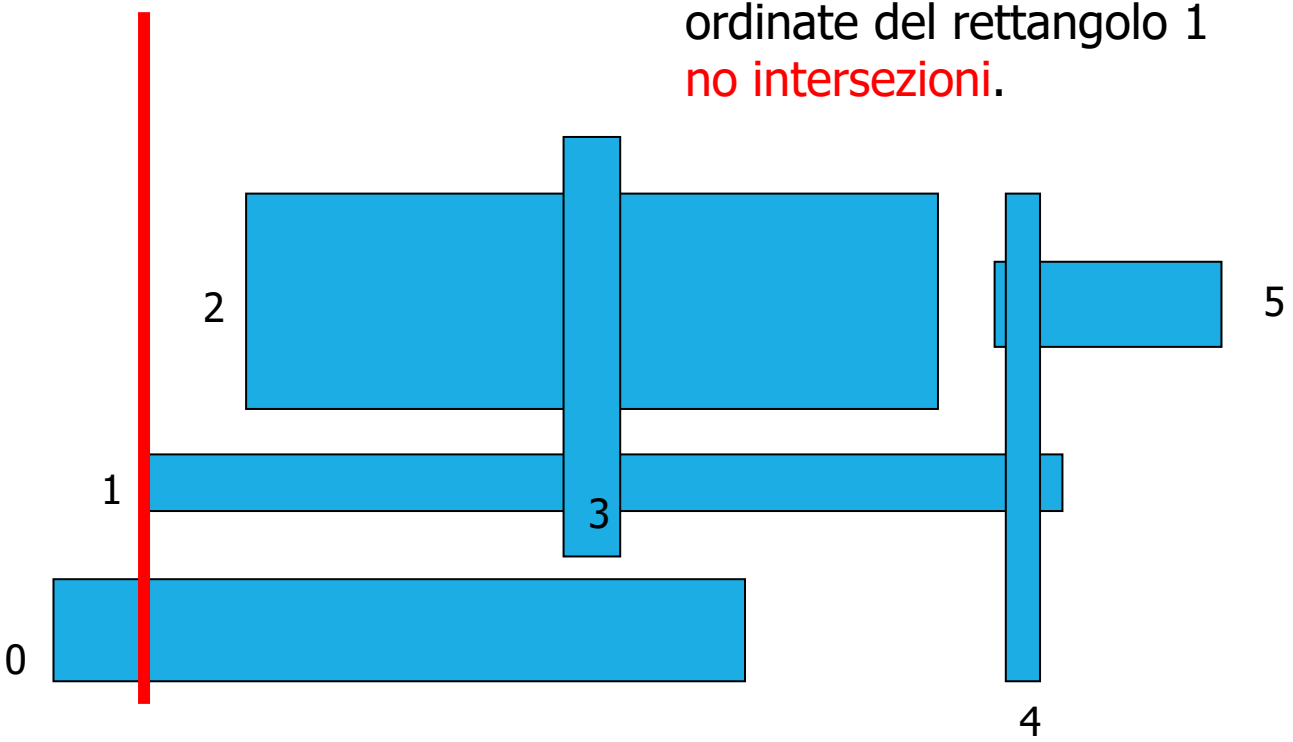
Complessità O(NlogN), applicabilità a VLSI e oltre:

- ordina i rettangoli per ascisse dell'estremo sinistro crescenti
- itera sui rettangoli per ascisse crescenti:
  - quando incontri l'estremo sinistro, inserisci in un I-BST l'intervallo delle ordinate e controlla l'intersezione
  - quando incontri l'estremo destro, rimuovi l'intervallo delle ordinate dall'I-BST.

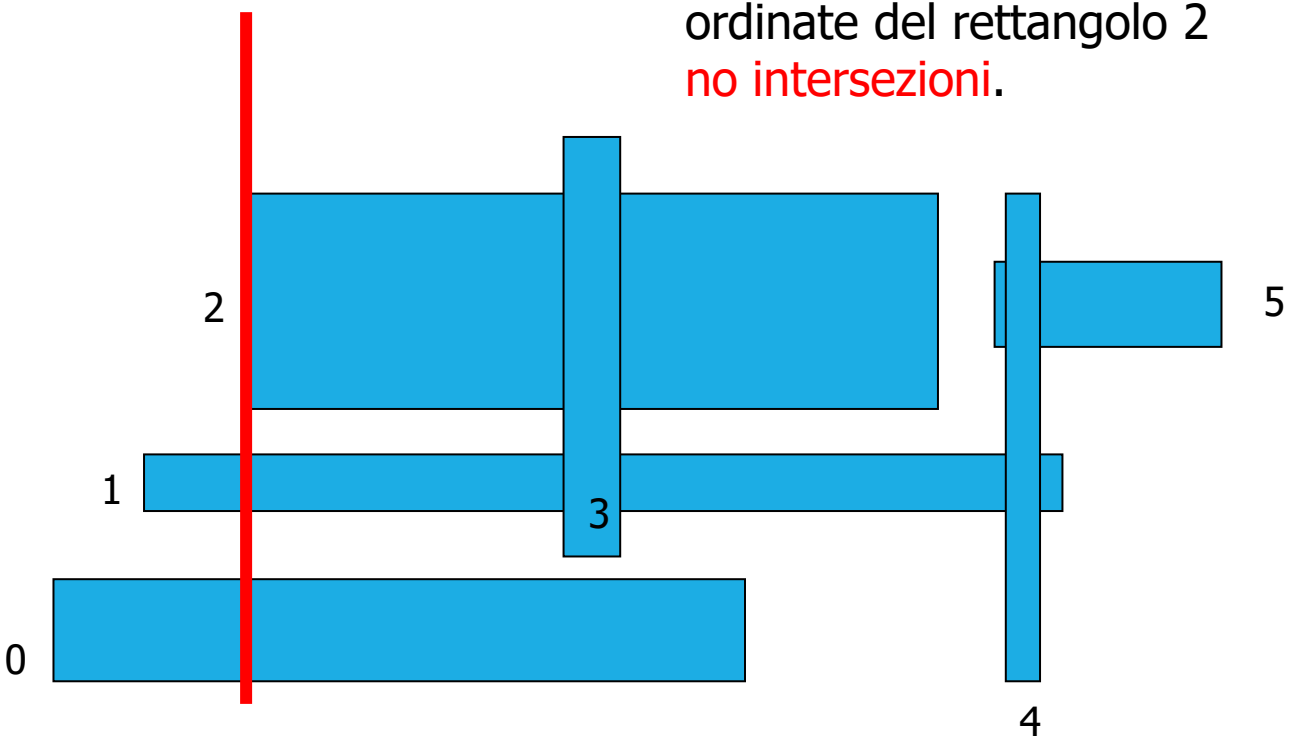
inserisci intervallo ordinate del rettangolo 0 no intersezioni.



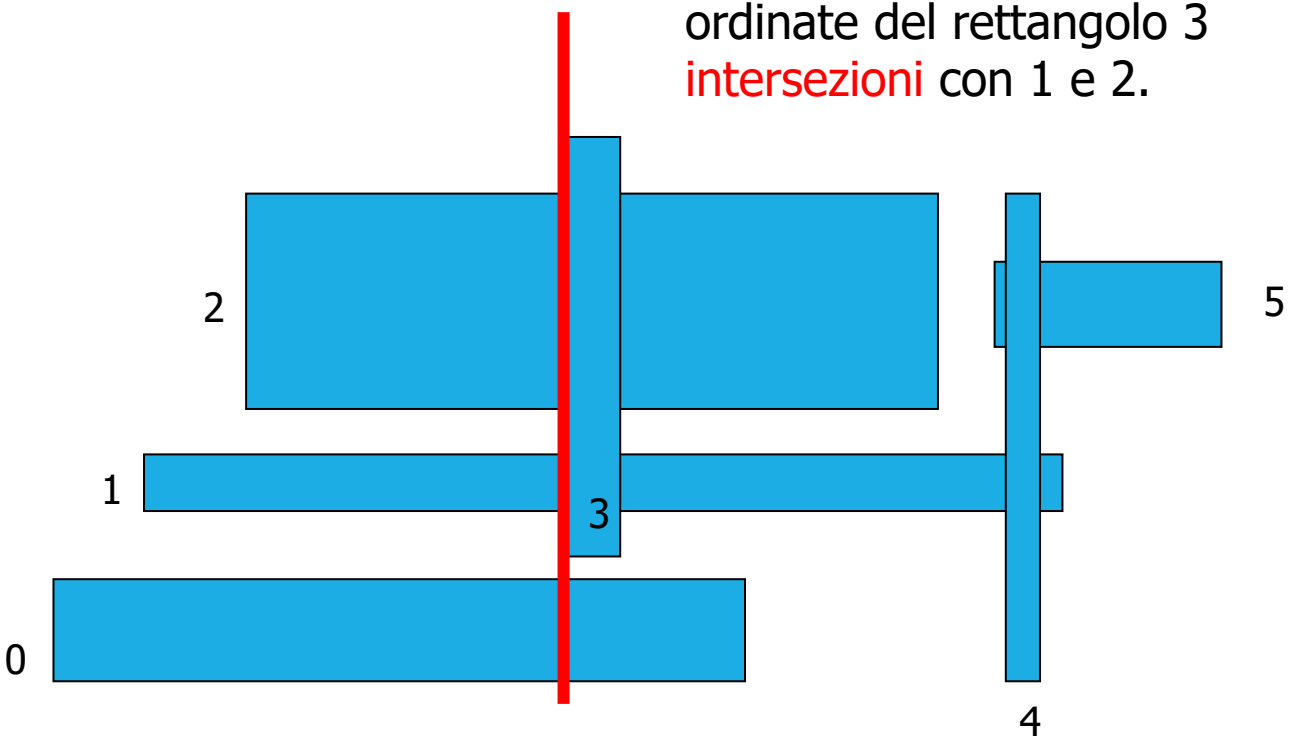
inserisci intervallo ordinate del rettangolo 1 no intersezioni.



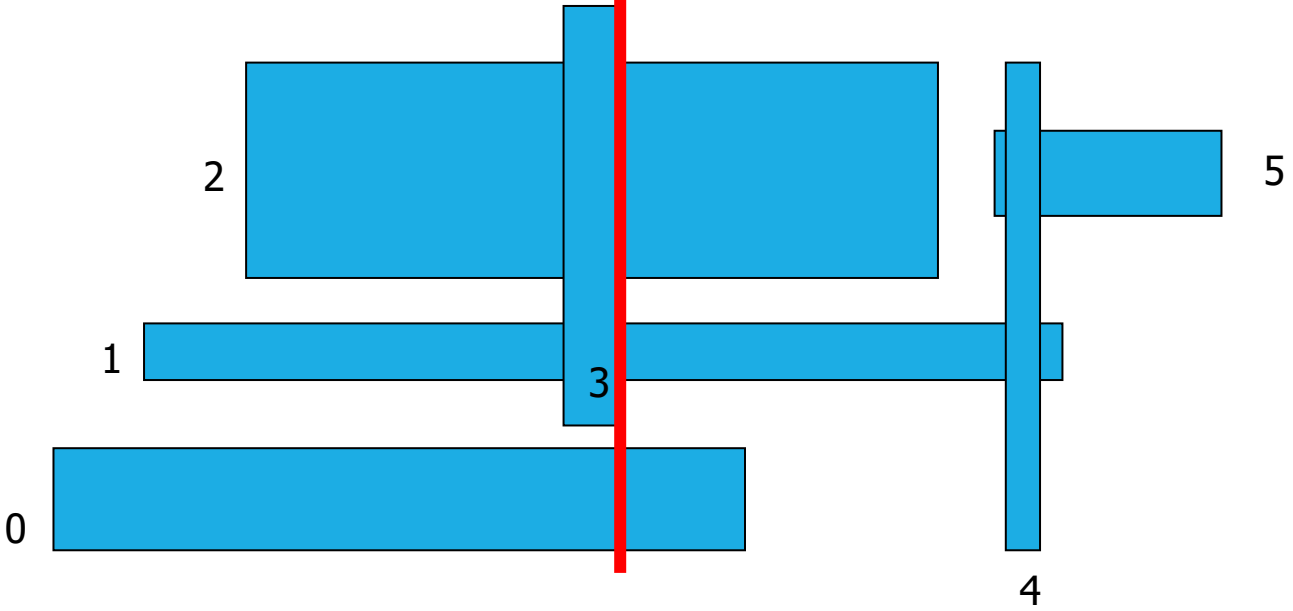
inserisci intervallo ordinate del rettangolo 2
no intersezioni.



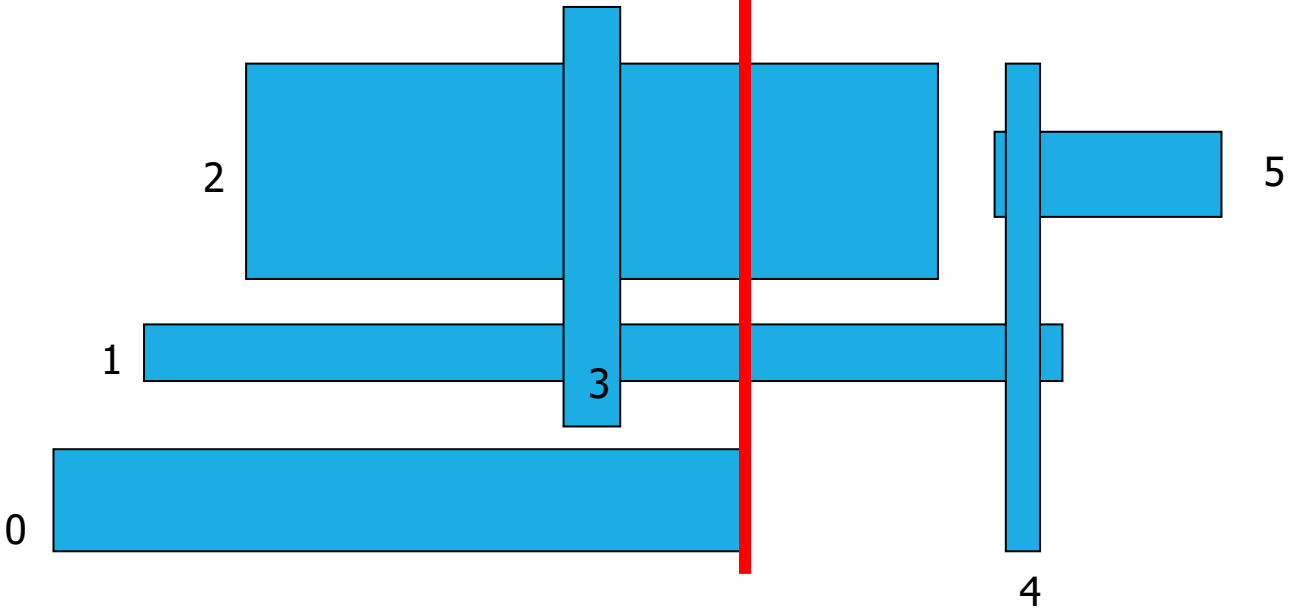
inserisci intervallo
ordinate del rettangolo 3
intersezioni con 1 e 2.

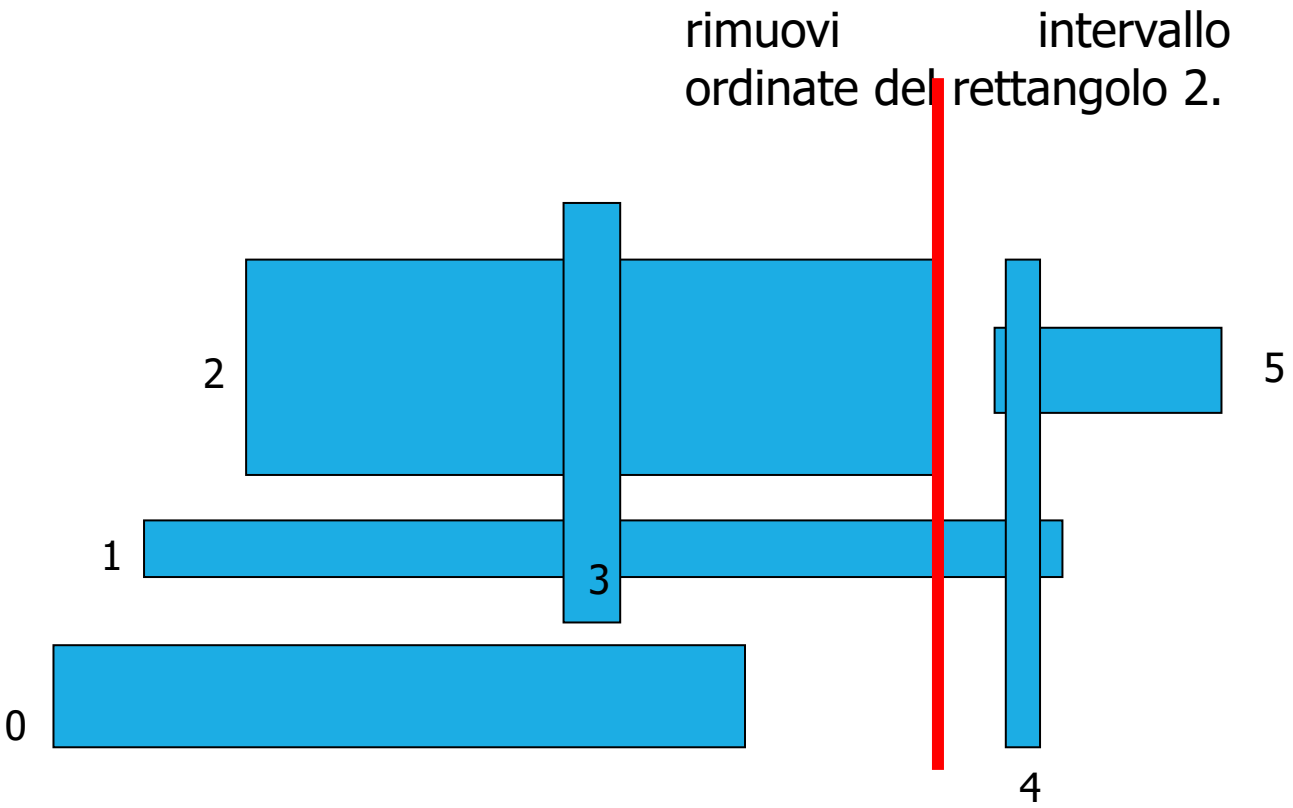


rimuovi intervallo
ordinate del rettangolo 3.

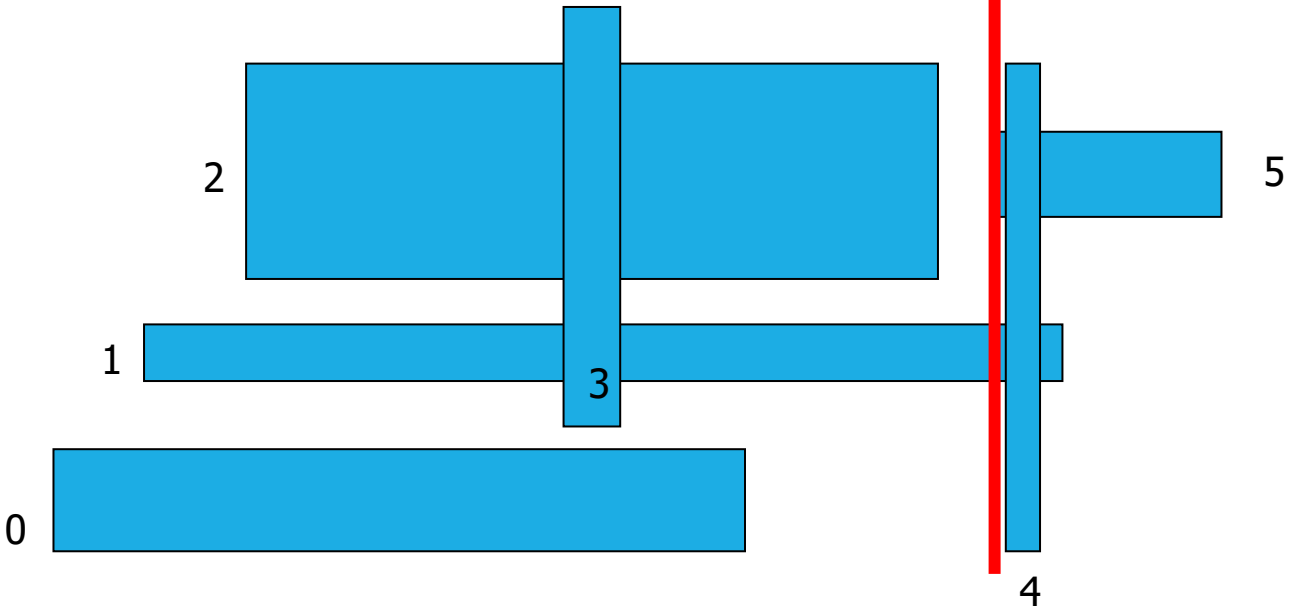


rimuovi intervallo
ordinate del rettangolo 0.

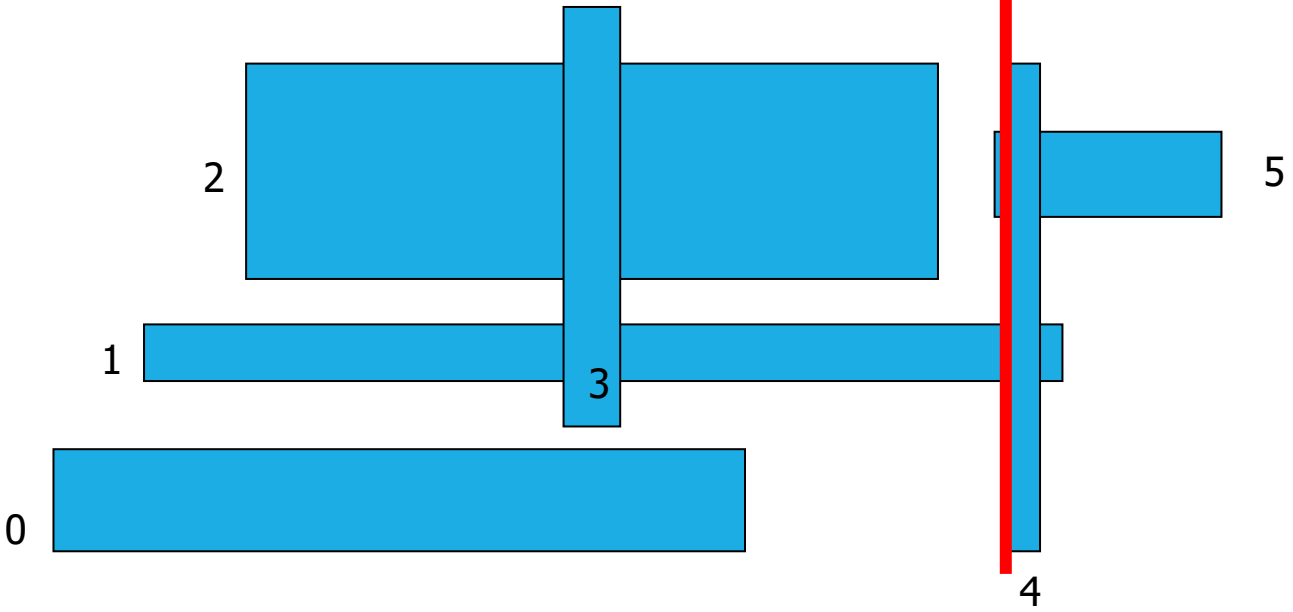



rimuovi intervallo
ordinate del rettangolo 2.
2
5
1
3
0
4

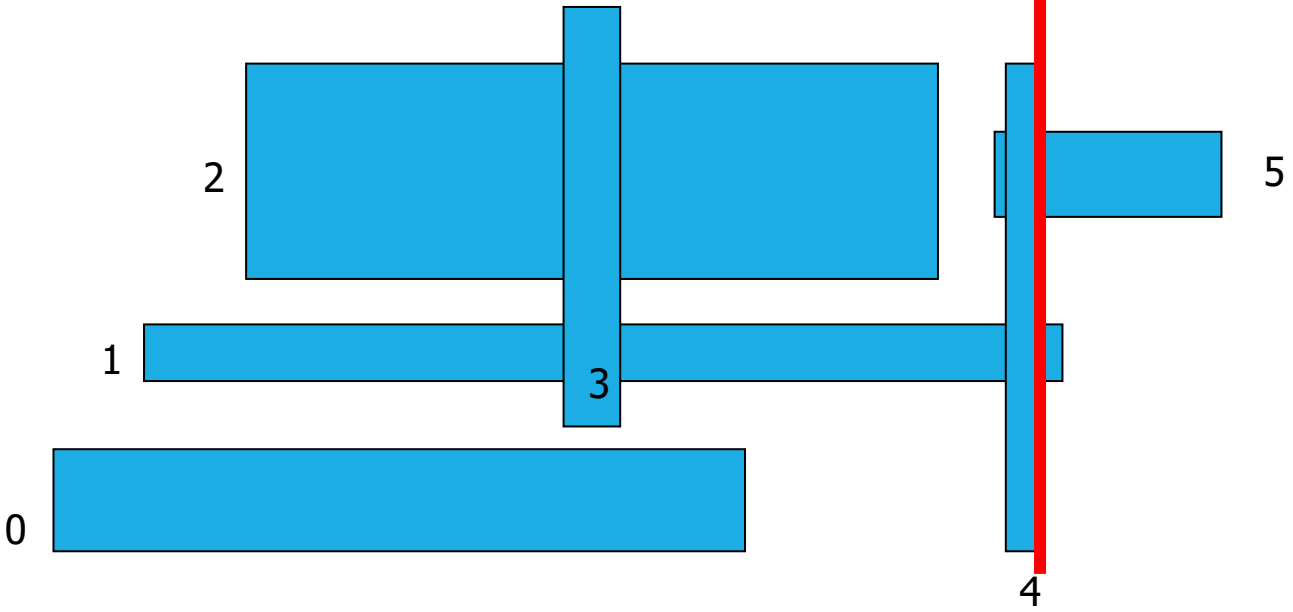
inserisci intervallo
ordinate del rettangolo 5
no intersezioni.



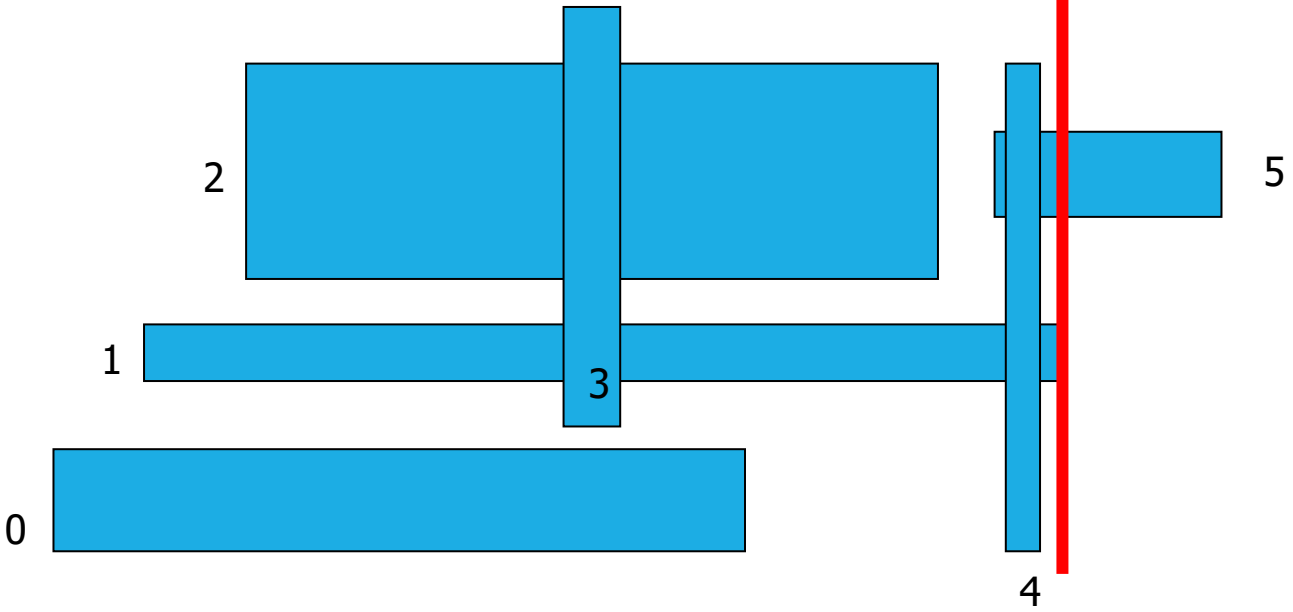
inserisci intervallo
ordinate del rettangolo 4
intersezioni con 1 e 5.



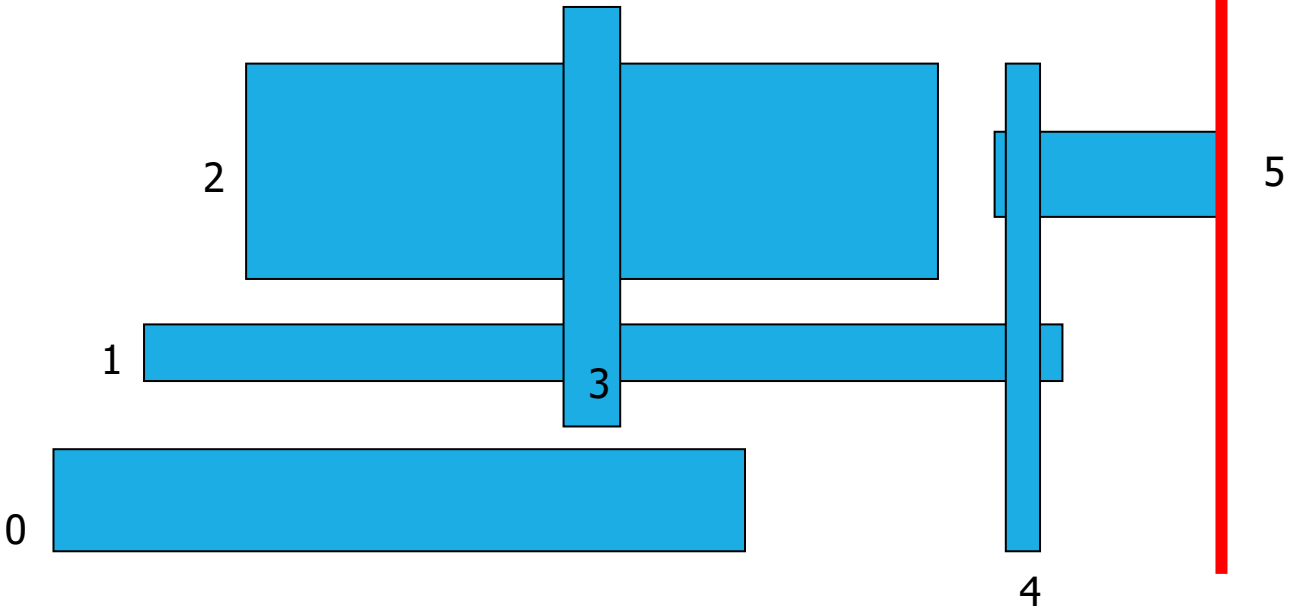
rimuovi intervallo
ordinate del rettangolo 4.



rimuovi intervallo
ordinate del rettangolo 1.



rimuovi intervallo
ordinate del rettangolo 5.



Analisi

Ordinamento: T(n) = O(NlogN)

Se l'IBST è bilanciato:

- ogni inserzione/cancellazione di intervallo o ricerca del primo intervallo che interseca uno dato costa T(n) = O(logN),
- la ricerca di tutti gli intervalli che intersecano un intervallo dato costa T(n) = O(RlogN) se R è il numero di intersezioni.

Riferimenti

- Alberi binari
  - Sedgewick 5.6, 5.7
- Binary Search Tree
  - Cormen 13.1, 13.2, 13.3
  - Sedgewick 12.5, 12.8, 12.9
- Order-statistic BST
  - Cormen 15.1
- Interval BST
  - Cormen 15.3