Lezione 9

Alberi binari di ricerca

Rossano Venturini

rossano.venturini@unipi.it

Pagina web del corso

http://didawiki.cli.di.unipi.it/doku.php/informatica/all-b/start

Esercizio 1

Tabelle Hash: inserimento

Scrivere un programma che legga da tastiera una sequenza di n interi **distinti** e li inserisca in una tabella hash di dimensione $2n$ posizioni utilizzando liste monodirezionali per risolvere eventuali conflitti.

Utilizzare la funzione hash $h(x) = ((ax + b) \ \% \ p) \ \% \ 2n$ dove p è il numero primo 999149 e a e b sono interi positivi minori di 10.000 scelti casualmente.

Una volta inseriti tutti gli interi, il programma deve stampare la lunghezza massima delle liste e il numero totale di conflitti.

Prima di scrivere il programma chiedersi perché la tabella ha dimensione $2n$ e non n.

Esercizio 1

```
typedef struct _node {
   struct _node *next;
   int value;
} node;

int hash(int x, int a, int b, int table_size) {
   int p = 999149;
   return ((a*x + b) % p) % table_size;
}
```

Esercizio 1

```
void insert_list(node** list, int x) {
  node* new = (node*)malloc(sizeof(node));
  new->value = x;
  new->next = *list;
  *list=new;
}


void insert(node** ht, int x, int a, int b, int table_size) {
  int index = hash(x, a, b, table_size);
  insert_list(&ht[index], x);
}


int len(node* list) {
  if (list==NULL) return 0;
  return 1+len(list->next);
}
```

Esercizio 2

Tabelle Hash: inserimento con rimozione dei duplicati

Scrivere un programma che legga da tastiera una sequenza di n interi **NON distinti** e li inserisca senza duplicati in una tabella hash di dimensione $2n$ posizioni utilizzando liste monodirezionali per risolvere eventuali conflitti.

Utilizzare la funzione hash $h(x) = ((ax + b) \quad \% \quad p) \quad \% \quad 2n$ dove p è il numero primo 999149 e a e b sono interi positivi minori di 10.000 scelti casualmente.

Una volta inseriti tutti gli interi, il programma deve stampare il numero totale di conflitti, la lunghezza massima delle liste e il numero di elementi distinti.

Esercizio 2

```
void insert_list(node** list, int x) {
  while (*list != NULL) {
    if ((*list)->value == x) return;
    list = &((*list)->next);
  }

  node* new = (node*)malloc(sizeof(node));
  new->value = x;
  new->next = NULL;
  *list=new;
}
```

Esercizio 3

Liste: cancellazione

Scrivere un programma che legga da tastiera una sequenza di n interi distinti e li inserisca in una lista monodirezionale. Successivamente il programma deve calcolare la media aritmetica dei valori della lista ed eliminare tutti gli elementi il cui valore è inferiore o uguale alla media, troncata all'intero inferiore. Ad esempio:

$$avg(1, 2, 4) = 7/3 = 2$$

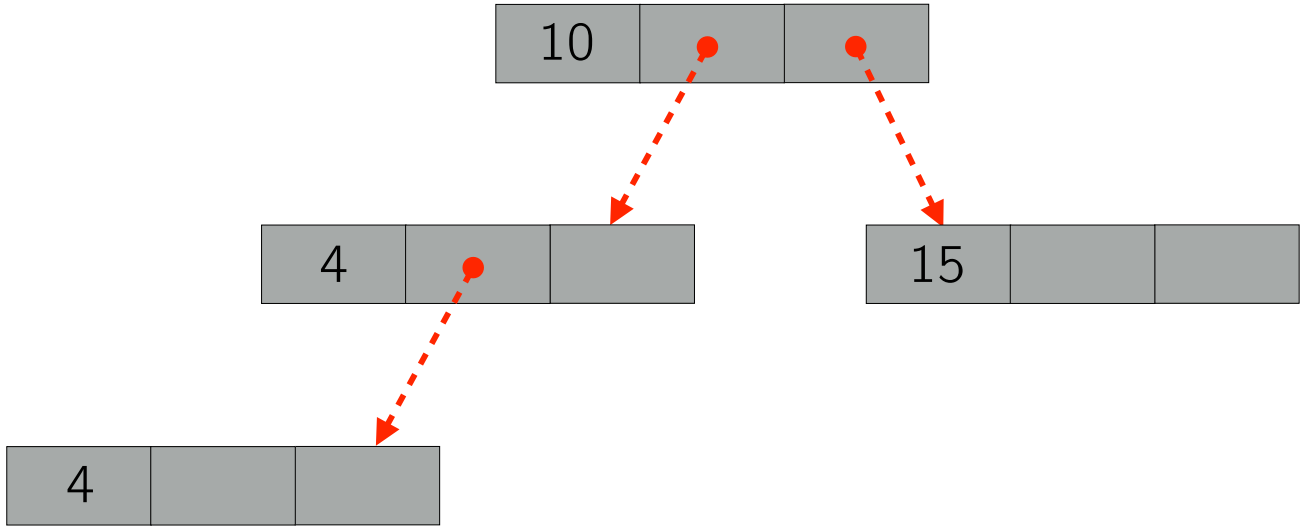
IMPORTANTE: Si abbia cura di liberare la memoria dopo ogni cancellazione.

Esercizio 3

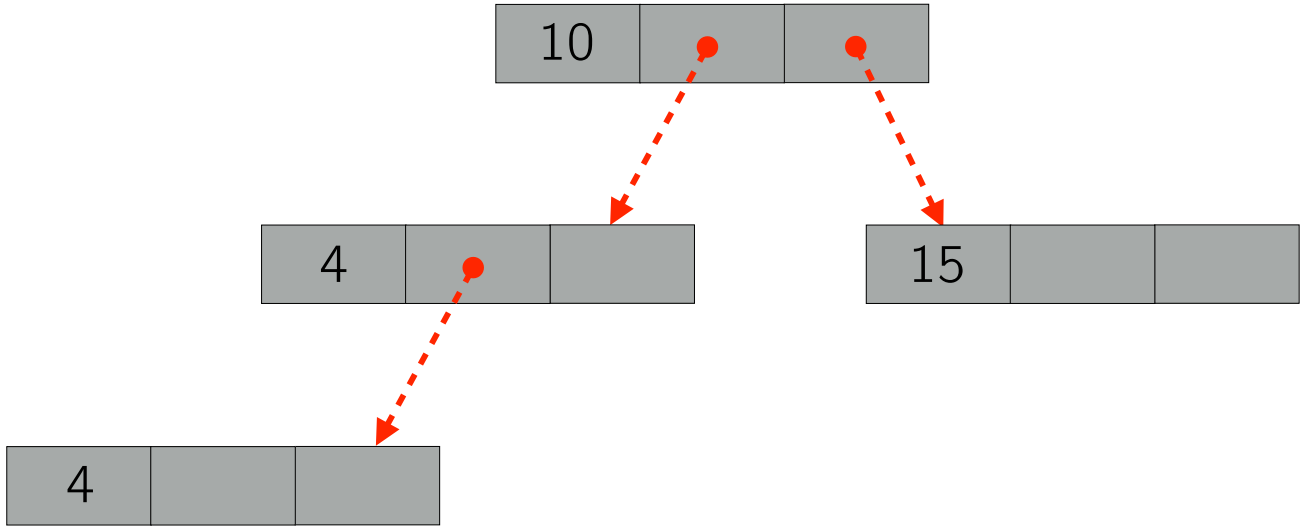
```
int list_average(node* list) {
  int sum = 0, count = 0;
  for (; list != NULL; list = list->next) {
    sum += list->value;
    ++count;
  }
  return (count > 0) ? (sum / count) : 0;
}

void delete_below(node** list, int x) {
  while (*list != NULL) {
    if ((*list)->value > x) {
      list = &(*list)->next;
    } else {
      node* to_delete = *list;
      *list = (*list)->next;
      free(to_delete);
    }
  }
}
```

Alberi binari di ricerca (ABR)

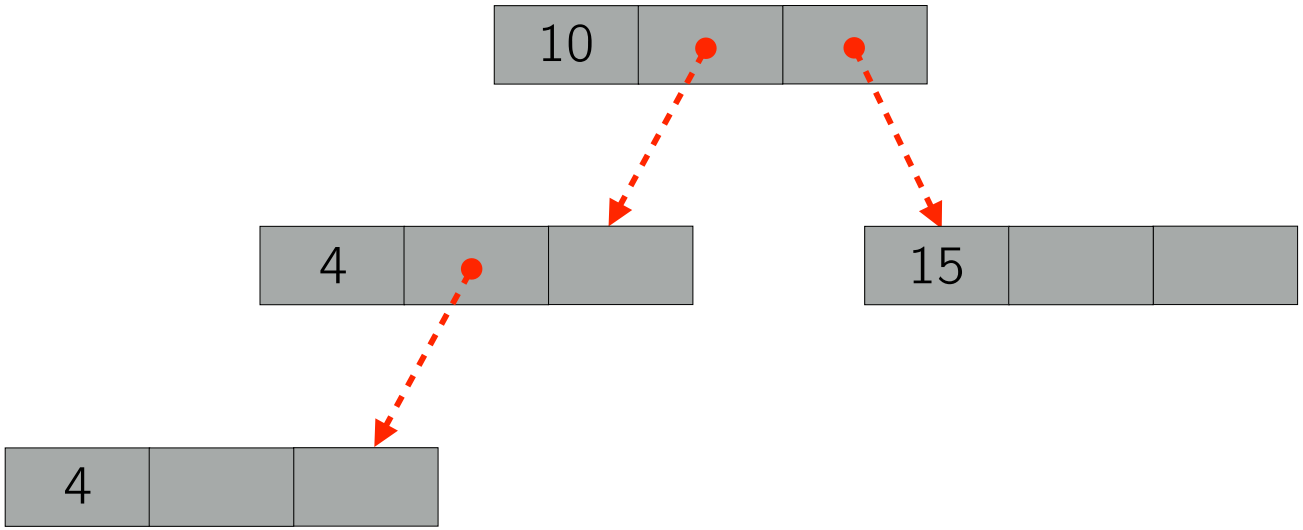


Alberi binari di ricerca (ABR)

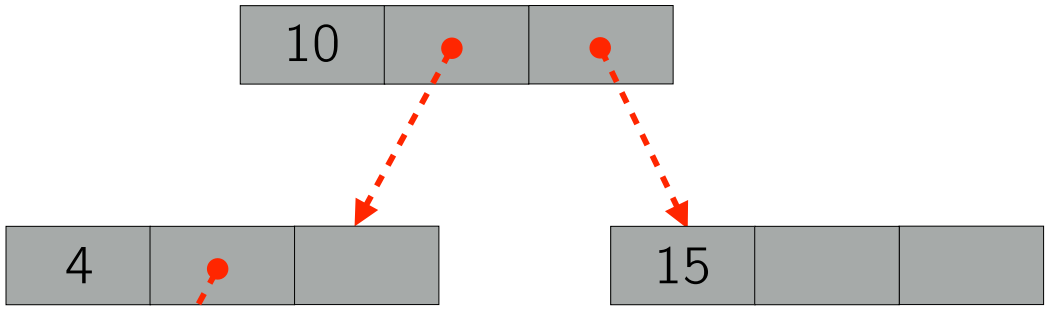



```
typedef struct _nodo {
   int key;
   struct _nodo *left;
   struct _nodo *right;
} nodo;
```

ABR: Ricerca

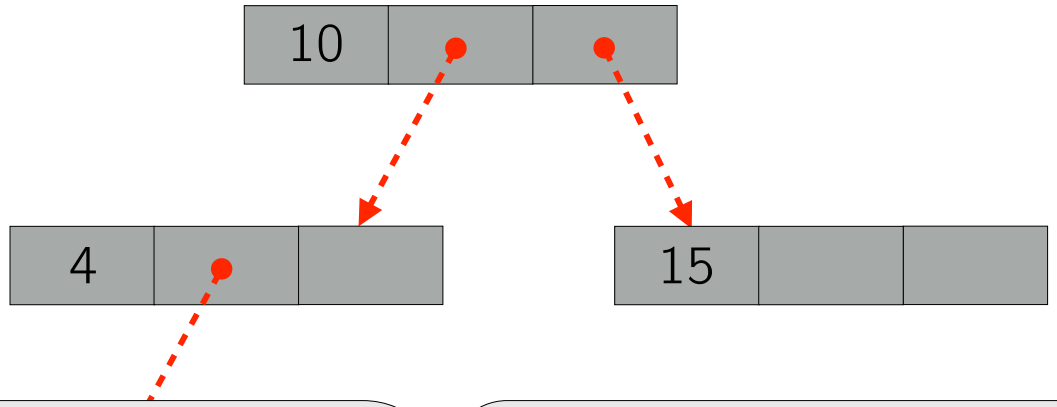


ABR: Ricerca




```
int cerca(nodo* t, int key) {
      int depth = 0;
      nodo* current = t;
      while (current != NULL) {
            if (key == current->key) return depth;
            if (key  >  current->key)
                  current = current->right;
            else
                  current = current->left;
            depth++;
      }
      return -1;
}
```

ABR: Ricerca

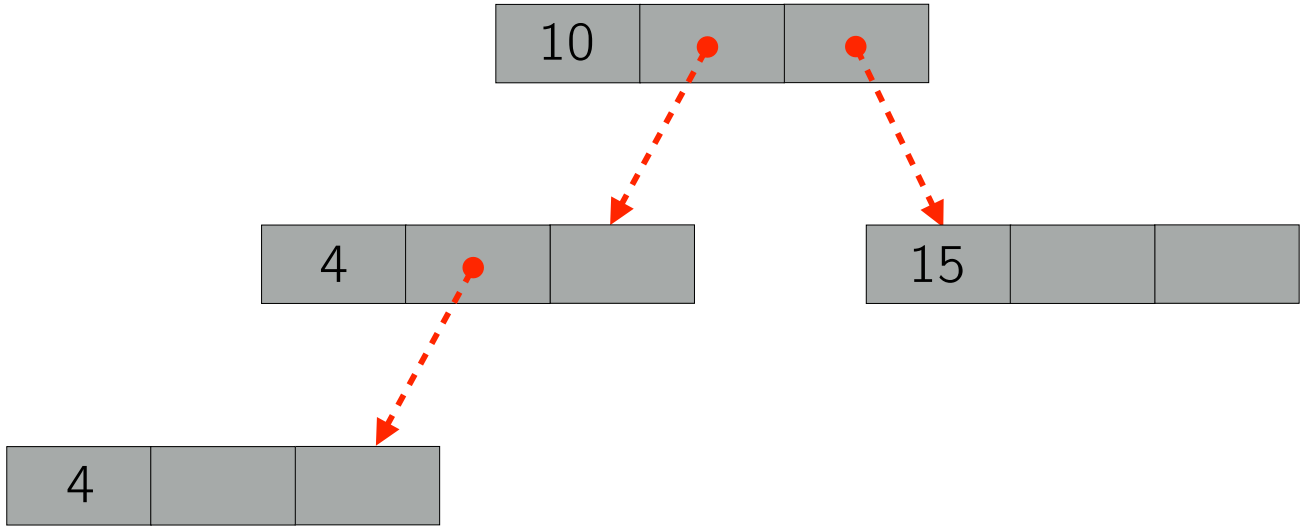



```
int cerca(nodo* t, int key) {
     int depth = 0;
     nodo* current = t;
     while (current != NULL) {
          if (key == current->key) return depth;
          if (key  >  current->key)
               current = current->right;
          else
               current = current->left;
          depth++;
     }
     return -1;
}
```

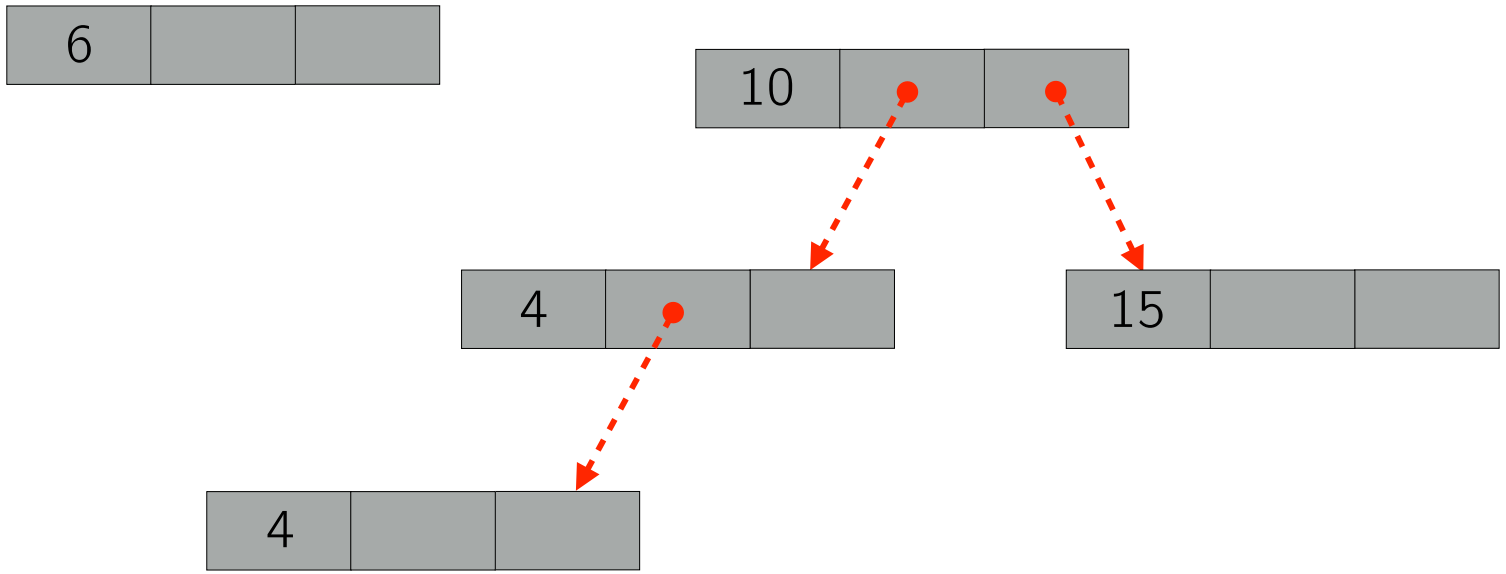
```
int cercaRic(nodo* t, int key) {
     if (t == NULL) return -1;
     if (key == t->key) return 0;
     int found = -1;
     if (key  >  t->key)
          found = cercaRic(t->right,key);
     else
          found = cercaRic(t->left,key);

     if (found >= 0) return 1 + found;
     else return -1;
}
```

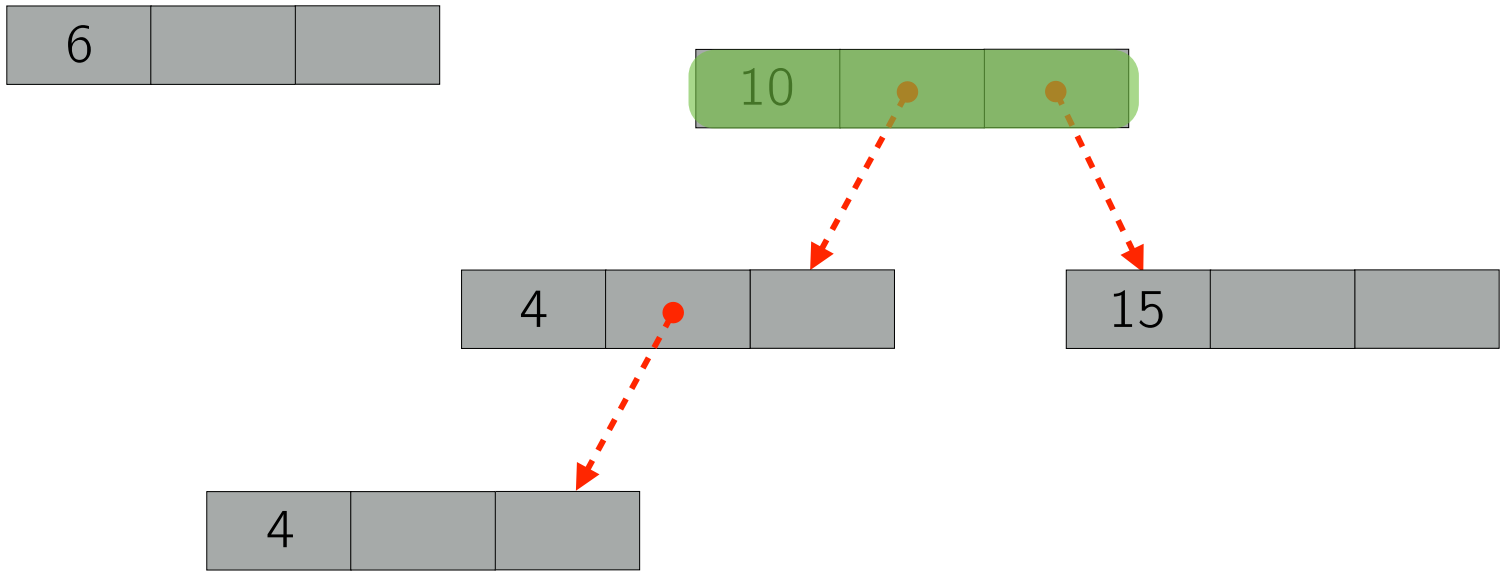
ABR: Inserimento



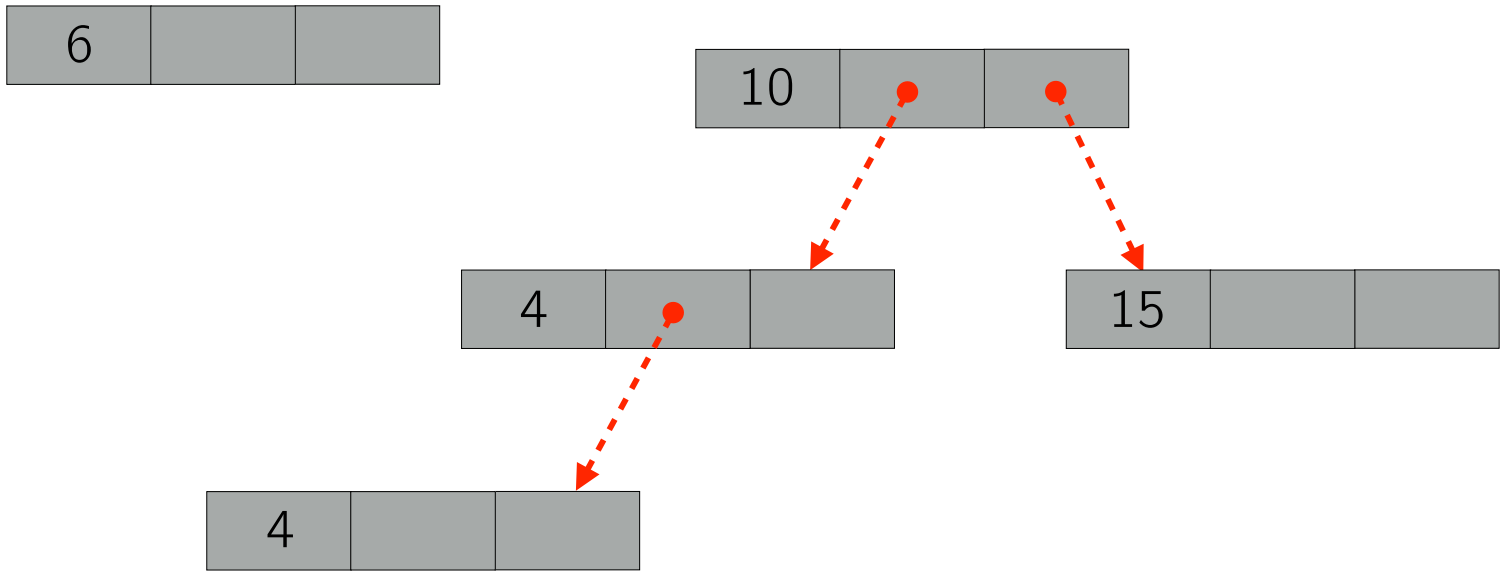
ABR: Inserimento



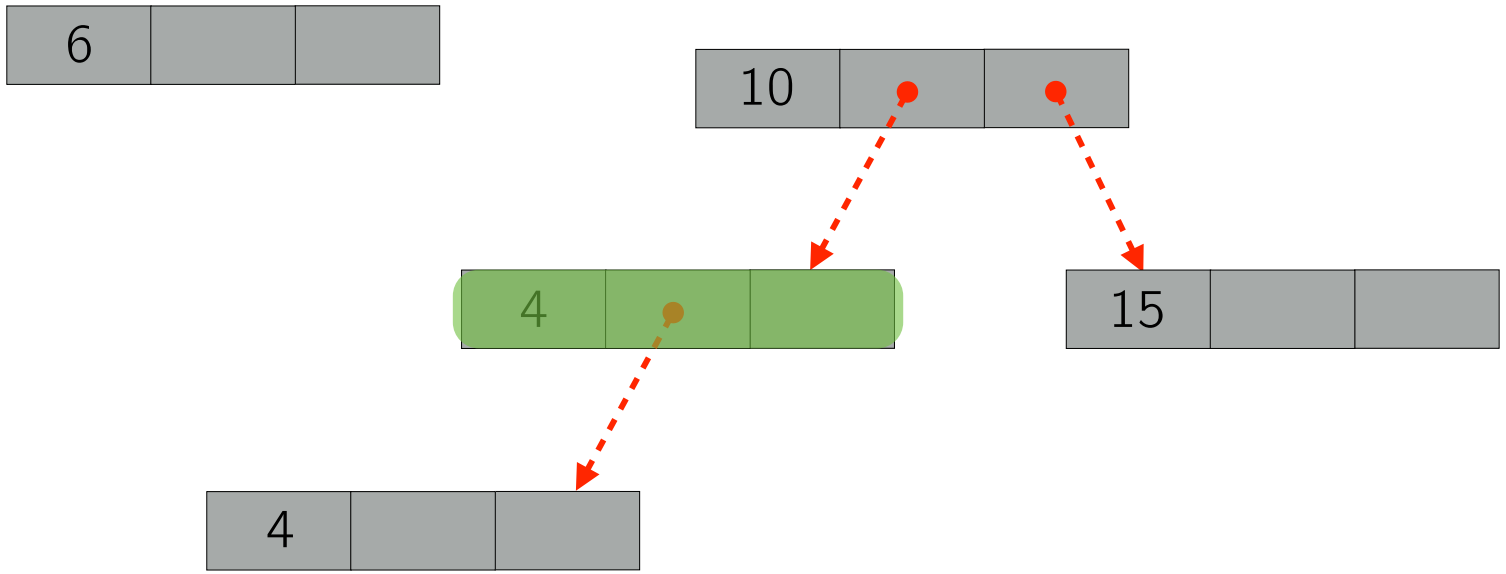
ABR: Inserimento



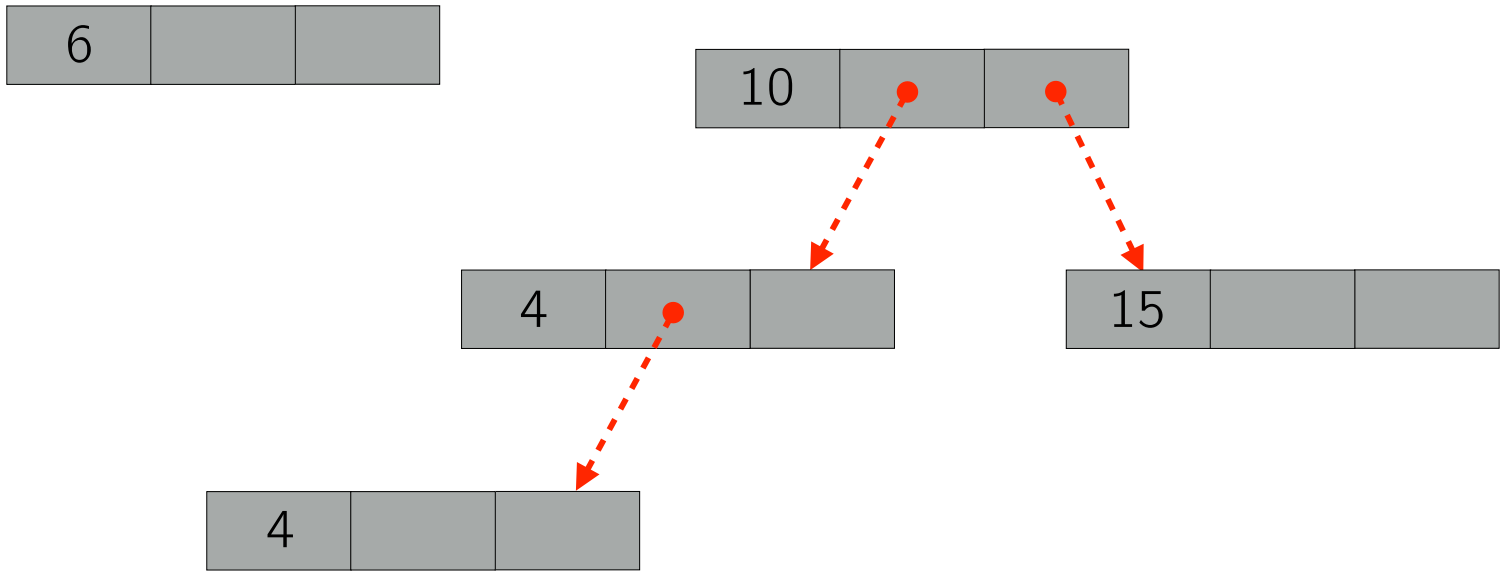
ABR: Inserimento



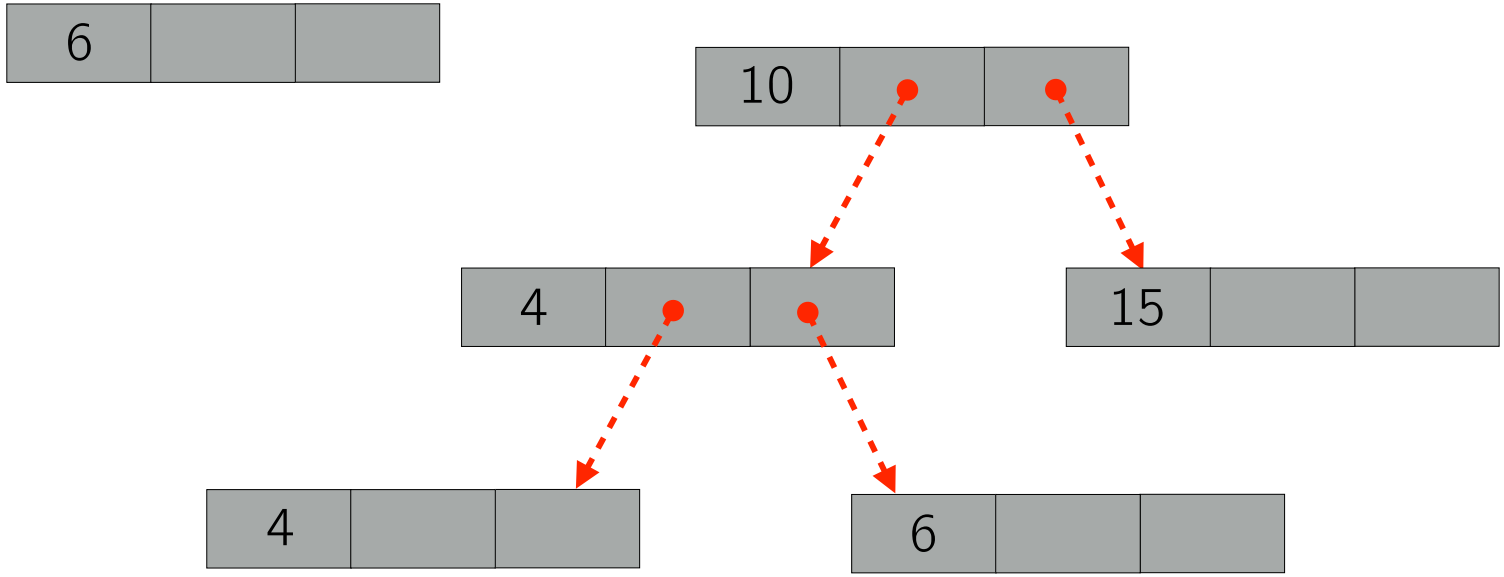
ABR: Inserimento



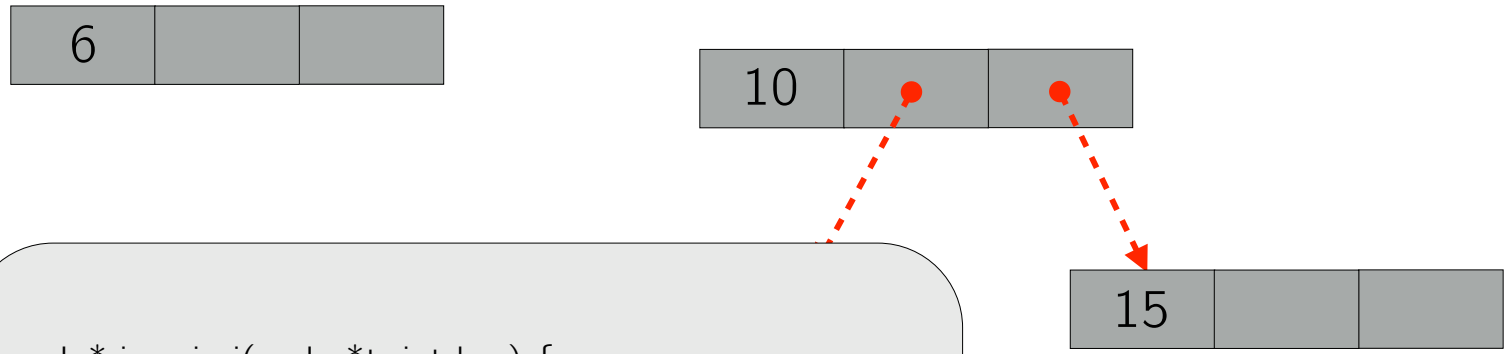
ABR: Inserimento



ABR: Inserimento

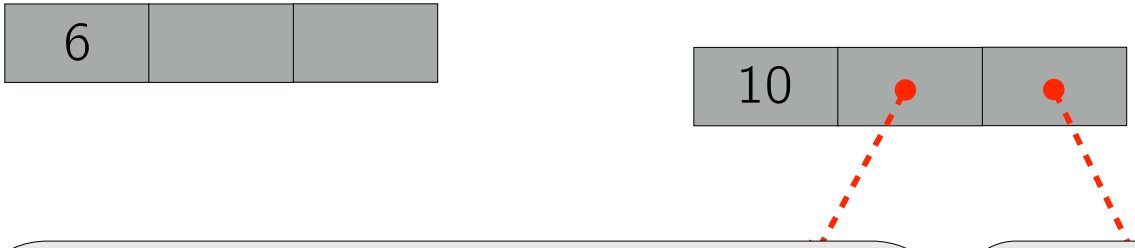


ABR: Inserimento




```
nodo* inserisci(nodo *t, int key) {
      nodo* new = (nodo *) malloc(sizeof(nodo));
      new->key = key;
      new->right = NULL; new->left = NULL;
      if (t == NULL) { return new; }
      nodo* parent;
      nodo* current = t;
      while (current != NULL) {
            parent = current;
            if (current->key < key)
                  current = current->right;
            else current = current->left;
      }
      if (parent->key < key) parent->right = new;
      else parent->left=new;
      return t;
}
```

ABR: Inserimento




```
nodo* inserisci(nodo *t, int key) {
      nodo* new = (nodo *) malloc(sizeof(nodo));
      new->key = key;
      new->right = NULL; new->left = NULL;
      if (t == NULL) { return new; }
      nodo* parent;
      nodo* current = t;
      while (current != NULL) {
            parent = current;
            if (current->key < key)
                  current = current->right;
            else current = current->left;
      }
      if (parent->key < key) parent->right = new;
      else parent->left=new;
      return t;
}
```

```
nodo* inserisciRic(nodo *t, int key) {
      if (t == NULL) {
         nodo* new = (nodo *) malloc(sizeof(nodo));
         new->key=key;
         new->left=NULL;
         new->right=NULL;
         return new;
      }

      if(t->key < key)
            t->right = inserisciRic(t->right, key);
      else
            t->left = inserisciRic(t->left, key);
      return t;
}
```

Esercizio 1

ABR: Ricerca

Scrivere un programma che legga da tastiera una sequenza di N interi distinti e li inserisca in un albero binario di ricerca (senza ribilanciamento).
Il programma entra poi in un ciclo infinito nel quale legge un intero i da tastiera e lo cerca nell'albero. Il ciclo termina solo se l'intero i inserito è strettamente minore di 0. Se i si trova nell'albero stampa la profondità alla quale l'elemento si trova (contando da 0), altrimenti stampa `NO`.

Esercizio 2

ABR: Visita

Scrivere un programma che legga da tastiera una sequenza di N interi distinti e li inserisca in un albero binario di ricerca (senza ribilanciamento).
Il programma deve visitare opportunamente l'albero e restituire la sua altezza.

Esercizio 3

ABR: Ordinamento

Scrivere un programma che legga da tastiera una sequenza di N interi distinti e li inserisca in un albero binario di ricerca (senza ribilanciamento).
Il programma deve quindi utilizzare un'opportuna visita dell'albero per stampare gli interi della sequenza in ordine non decrescente.

Esercizio 4
ABR: Confronto (prova del 11/06/2011)

Scrivere un programma che riceve in input due sequenze di N interi positivi, dalle quali devono essere costruiti due alberi binari di ricerca NON bilanciati (un albero per sequenza).
Al programma viene data una chiave intera K. Si può assumere che l'intero K sia presente in entrambe le sequenze. Il programma deve verificare che le sequenze di chiavi incontrate nel cammino che dalla radice porta al nodo con chiave K nei due alberi coincidano.

Esercizio 5

Albero Ternario (prova del 01/02/2012)

Scrivere un programma che riceva in input una sequenza di N interi positivi e costruisca un albero **ternario** di ricerca **non** bilanciato. L'ordine di inserimento dei valori nell'albero deve coincidere con quello della sequenza.

Ogni nodo in un albero ternario di ricerca può avere fino a tre figli: figlio sinistro, figlio centrale e figlio destro. L'inserimento di un nuovo valore avviene partendo dalla radice dell'albero e utilizzando la seguente regola. Il valore da inserire viene confrontato con la chiave del nodo corrente. Ci sono tre possibili casi in base al risultato del confronto:

1. se il valore è minore della chiave del nodo corrente, esso viene inserito ricorsivamente nel sottoalbero radicato nel figlio sinistro;

2. se il valore è **divisibile** per la chiave del nodo corrente, esso viene inserito ricorsivamente nel sottoalbero radicato nel figlio centrale;

3. in ogni altro caso il valore viene inserito ricorsivamente nel sottoalbero radicato nel figlio destro.

Il programma deve stampare il numero di nodi dell'albero che hanno **tre** figli.

Esercizio 6 (Facoltativo)

La ditta Serling Enterprises compra lunghe barre d'acciaio e le taglia in barrette più piccole, che poi rivende. L'operazione di taglio non comporta costi aggiuntivi. La direzione della Serling Enterprises vuole scoprire la maniera migliore per tagliare le barre, in maniera da massimizzare il guadagno.

Detta n la lunghezza in pollici di una barra da tagliare, supponiamo di conoscere, per $i = 1, 2, \ldots, n$, il prezzo p_i in dollari a cui l'azienda vende una barra di lunghezza i pollici. Le lunghezze delle barre sono sempre uguali ad un numero intero di pollici. Un esempio è dato dalla tabella seguente:

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Data una barra di lunghezza n pollici e una tabella dei prezzi p_i per $i = 1, 2, \ldots, n$, si vuole determinare una maniera di tagliare la barra in modo da raggiungere il profitto massimo r_n ottenibile tagliando la barra e vendendone i pezzi. Si noti che se il prezzo p_n di una barra di lunghezza n è sufficientemente grande, allora una soluzione ottima potrebbe non richiedere alcun taglio. Si noti inoltre che in generale esistono più soluzioni ottime, tutte ovviamente caratterizzate dallo stesso valore r_n.

Consideriamo il caso in cui $n = 4$. La figura 1 mostra tutti i modi possibili di tagliare una barra di 4 pollici, incluso il caso che non prevede tagli. Facendo riferimento alla tabella dei prezzi precedente, sopra ciascuna immagine sono mostrati i prezzi delle barrette risultanti. Si vede facilmente che tagliare la barra in due barrette lunghe 2 pollici produce il guadagno massimo pari a $p_2 + p_2 = 5 + 5 = 10$ dollari.

Puzzle

Griglia infetta

Abbiamo una griglia 8×8 e la possibilità di infettare 7 delle sue celle. Ad ogni intervallo di tempo, vengono infettate tutte le celle che hanno almeno due lati infetti. Il processo termina non appena non sono più possibili nuove infezioni. Mostrare come scegliere le prime 7 celle in modo che alla fine del processo l'intera griglia sia infetta, o dimostrare che una tale configurazione non può esistere.