

Lezione 10

Tabelle Hash

Rossano Venturini

rossano.venturini@unipi.it

Pagina web del corso

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

Esercizio 1

Lista monodirezionale

Scrivere un programma che legga da tastiera una sequenza di n interi e li inserisca in una lista nell'ordine di immissione. La lista deve essere **monodirezionale**.

Il programma deve stampare il contenuto della lista percorrendola in ordine inverso.

Esercizio 1

```
typedef struct _item {  
    struct _item* next;  
    int value;  
} item;
```

```
item* create_list(int value) {  
    item* l = (item*)malloc(sizeof(item));  
    l->value = value;  
    l->next = NULL;  
    return l;  
}
```

```
item* append(item* last_elem, int value) {  
    last_elem->next = (item*) malloc(sizeof(item));  
    last_elem->next->value = value;  
    last_elem->next->next = NULL;  
    return last_elem->next;  
}
```

Esercizio 1

```
void print_list(item* l) {  
    if (l == NULL) return;  
    printf("%d\n", l->value);  
    print_list(l->next);  
    return;  
}
```

```
void print_reversed_list(item* l) {  
    if (l == NULL) return;  
    print_reversed_list(l->next);  
    printf("%d\n", l->value);  
    return;  
}
```

Esercizio 1

```
int main(void) {  
    ...  
    l = create_list(val);  
    next = l; len--;  
    while (len > 0) {  
        scanf("%d", &val);  
        next = append(next, val);  
        len--;  
    }  
    print_reversed_list(l);  
  
    return 0;  
}
```

Esercizio 2

Lista bidirezionale

Scrivere un programma che legga da tastiera una sequenza di n interi e li inserisca in una lista nell'ordine di immissione. La lista deve essere **bidirezionale**.

Il programma deve stampare il contenuto della lista percorrendola in ordine inverso.

Esercizio 2

```
typedef struct _node {  
    struct _node* next;  
    struct _node* prev;  
    int value;  
} item;
```

```
list create_list(int value) {  
    item* l = (item*)malloc(sizeof(item));  
    l->value = value;  
    l->next = NULL;  
    l->prev = NULL;  
    return l;  
}
```

```
item* append(item* last_elem, int value) {  
    last_elem->next = (item*)malloc(sizeof(item));  
    last_elem->next->value = value;  
    last_elem->next->next = NULL;  
    last_elem->next->prev = last_elem;  
    return last_elem->next;  
}
```

Esercizio 2

```
void print_list(item* l) {  
    if (l == NULL) return;  
    printf("%d\n", l->value);  
    print_list(l->next);  
    return;  
}
```

```
void print_reversed_list(item* last_elem) {  
    while (last_elem != NULL) {  
        printf("%d\n", last_elem->value);  
        last_elem = last_elem->prev;  
    }  
}
```

Tabelle Hash

Soluzione standard per il problema del dizionario

Tabelle Hash

Soluzione standard per il problema del dizionario

Mantenere un insieme $S \subseteq U$ e fornire le seguenti operazioni

Insert(x): inserisce x in S

Delete(x): rimuove x da S

Search(x): determina se x è in S

Tabelle Hash

Soluzione standard per il problema del dizionario

Mantenere un insieme $S \subseteq U$ e fornire le seguenti operazioni

Insert(x): inserisce x in S

Delete(x): rimuove x da S

Search(x): determina se x è in S

Vedremo Tabelle hash con gestione dei conflitti con liste di adiacenza

Tabelle Hash

Tabelle Hash

$S = \{5, 10, 15, 20, 22, 25, 33\}$

Tabelle Hash

$S = \{5, 10, 15, 20, 22, 25, 33\}$

$h: S \rightarrow |T|$

$h(5) = 1$

$h(10) = h(22) = h(33) = 3$

$h(15) = h(20) = 7$

$h(25) = 9$

Tabelle Hash

$S = \{5, 10, 15, 20, 22, 25, 33\}$

$h: S \rightarrow |T|$

$h(5) = 1$

$h(10) = h(22) = h(33) = 3$

$h(15) = h(20) = 7$

$h(25) = 9$



Tabelle Hash

$S = \{5, 10, 15, 20, 22, 25, 33\}$

$h: S \rightarrow |T|$

$h(5) = 1$

$h(10) = h(22) = h(33) = 3$

$h(15) = h(20) = 7$

$h(25) = 9$

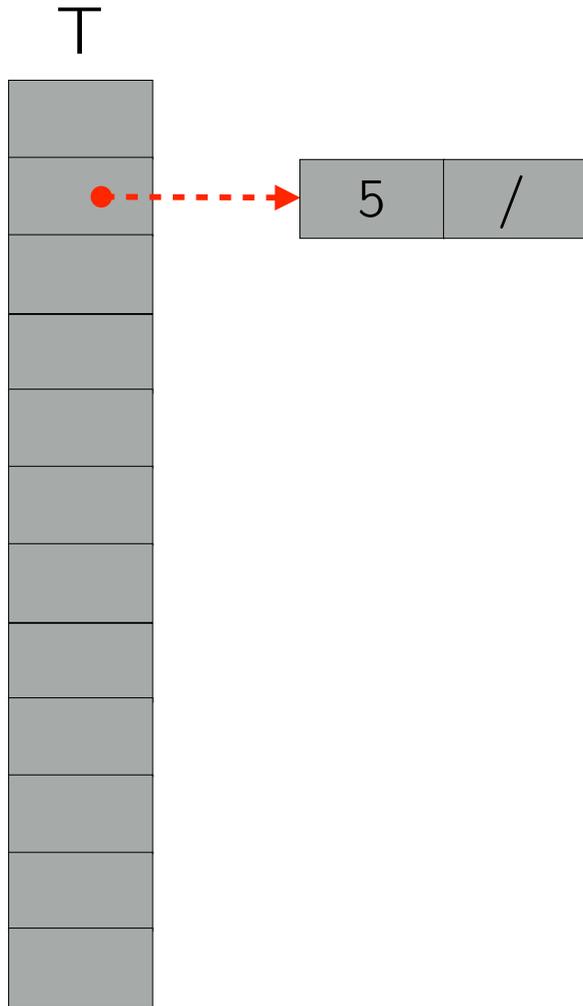


Tabelle Hash

$S = \{5, 10, 15, 20, 22, 25, 33\}$

$h: S \rightarrow |T|$

$h(5) = 1$

$h(10) = h(22) = h(33) = 3$

$h(15) = h(20) = 7$

$h(25) = 9$

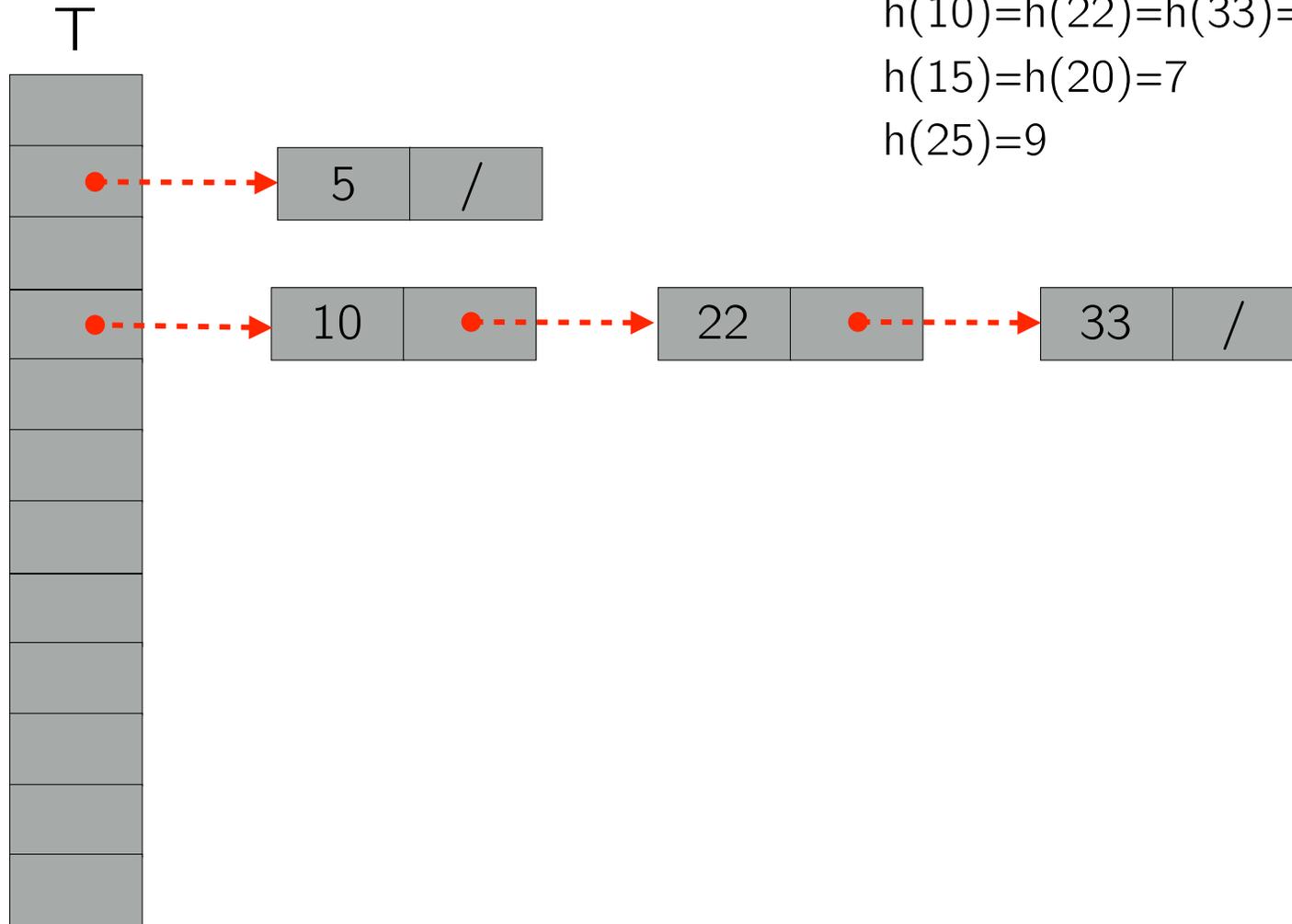


Tabelle Hash

$S = \{5, 10, 15, 20, 22, 25, 33\}$

$h: S \rightarrow |T|$

$h(5) = 1$

$h(10) = h(22) = h(33) = 3$

$h(15) = h(20) = 7$

$h(25) = 9$

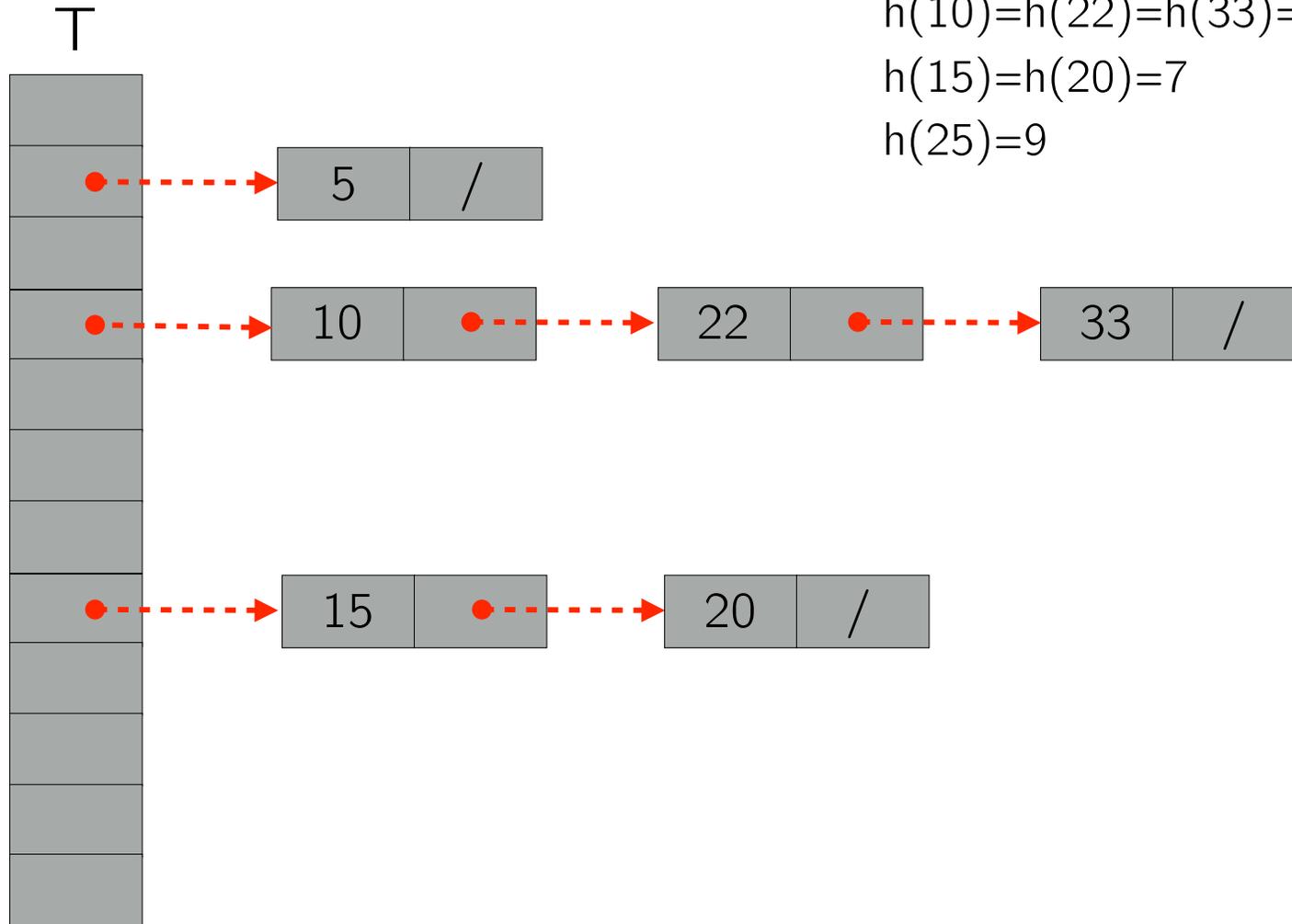


Tabelle Hash

$S = \{5, 10, 15, 20, 22, 25, 33\}$

$h: S \rightarrow |T|$

$h(5) = 1$

$h(10) = h(22) = h(33) = 3$

$h(15) = h(20) = 7$

$h(25) = 9$

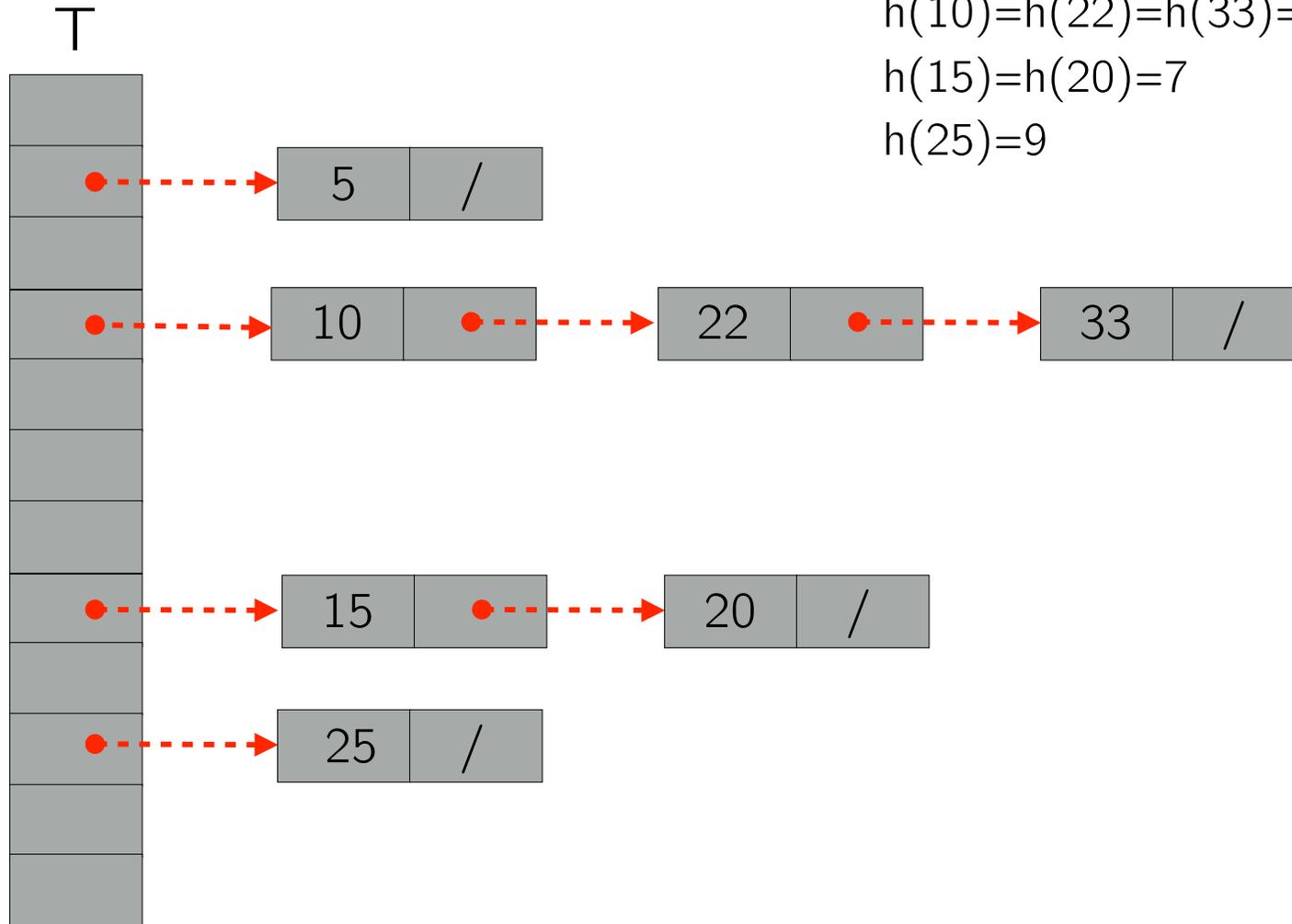


Tabelle Hash

$S = \{5, 10, 15, 20, 22, 25, 33\}$

$h: S \rightarrow |T|$

$h(5) = 1$

$h(10) = h(22) = h(33) = 3$

$h(15) = h(20) = 7$

$h(25) = 9$

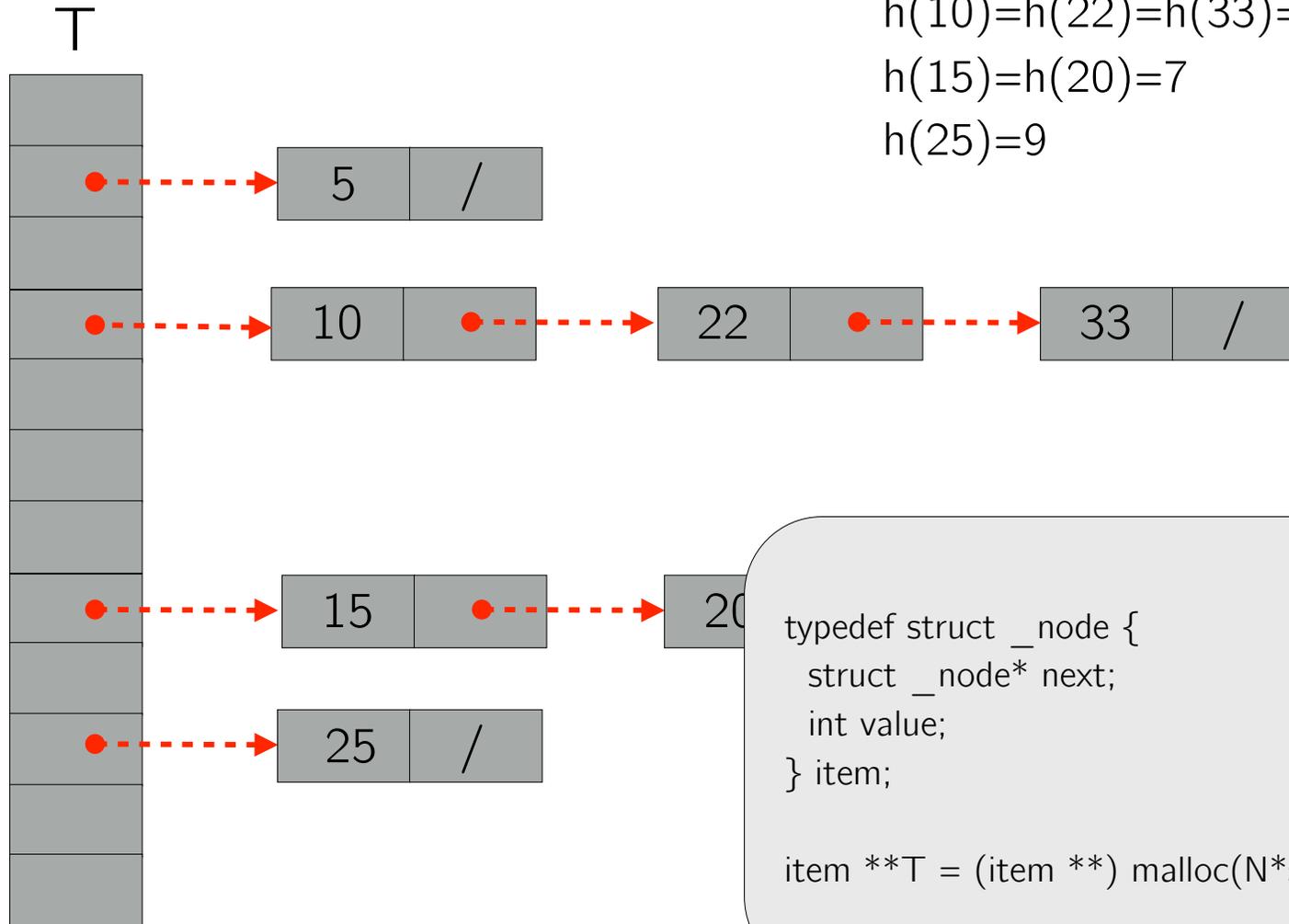


Tabelle Hash

$S = \{5, 10, 15, 20, 22, 25, 33\}$

$h: S \rightarrow |T|$

$h(5) = 1$

$h(33) = 3$

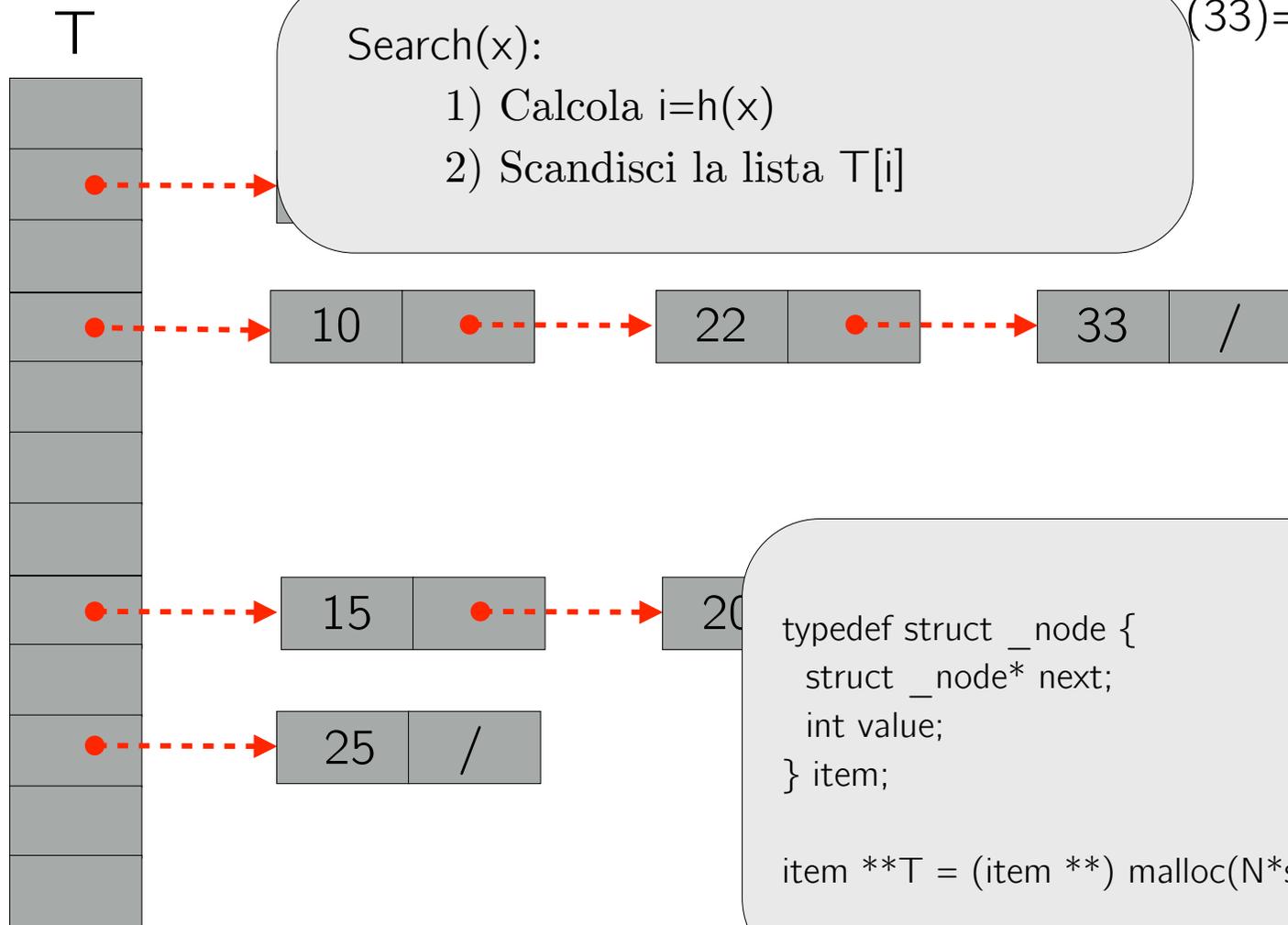


Tabelle Hash

$S = \{5, 10, 15, 20, 22, 25, 33\}$

$h: S \rightarrow |T|$

$h(5) = 1$

$h(33) = 3$



Search(x):
1) Calcola $i = h(x)$
2) Scandisci la lista $T[i]$

Insert(x):
1) Calcola $i = h(x)$
2) Inserisci x in testa/coda a $T[i]$



```
typedef struct _node {  
    struct _node* next;  
    int value;  
} item;  
  
item **T = (item **) malloc(N*sizeof(item *));
```

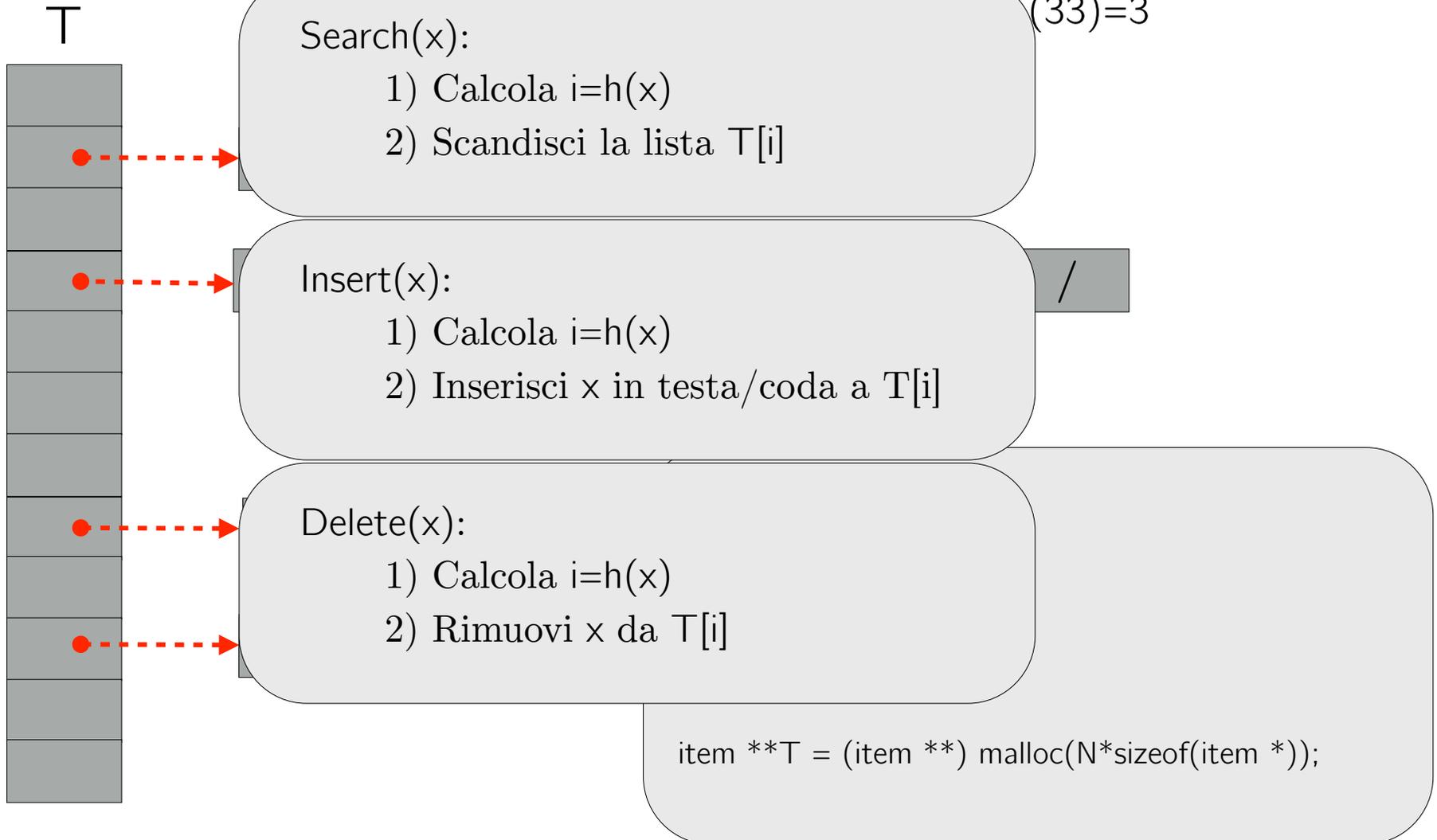
Tabelle Hash

$S = \{5, 10, 15, 20, 22, 25, 33\}$

$h: S \rightarrow |T|$

$h(5) = 1$

$h(33) = 3$



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 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 $h(x) = ((ax + b) \% p) \% 2n$ 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 il numero totale di conflitti, la lunghezza massima delle liste e il numero di elementi distinti.

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:

$$\text{avg}(1, 2, 4) = 7/3 = 2$$

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

Esercizio 4

Liste: Somme suffisse

Scrivere un programma che legga da tastiera una sequenza di n interi maggiori di 0 e li inserisca in una lista nell'ordine di immissione. La lista deve essere monodirezionale.

Successivamente il programma deve sostituire il valore di ciascun elemento con la somma dei valori degli elementi che lo seguono nella lista.

Suggerimento: si utilizzi la ricorsione per ottenere la somma ad ogni passo.

Puzzle

Orco e hobbit

Un orco ha rapito N hobbit e gli propone: "Domani mattina metterò un cappello a ciascuno di voi. Ogni cappello sarà etichettato con un numero da 0 a $N - 1$, doppianti sono possibili. Ogni hobbit potrà vedere il numero sui cappelli degli altri ma non il suo. Quando suonerò la campanella, tutti gli hobbit dovranno simultaneamente dire un numero (potenzialmente diverso per ogni hobbit). Se almeno un hobbit sarà in grado di indovinare il suo numero, tutti gli hobbit saranno liberati." La sera stessa gli hobbit si incontrano e individuano una strategia che è **sempre** vincente, quale?

Per $N = 2$ la strategia è relativamente facile mentre la soluzione per N arbitrario è decisamente più difficile.