Lezione 8

Liste

Rossano Venturini

rossano.venturini@unipi.it

Pagina web del corso

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

Esercizio 1

Prova del 18/05/2009

Scrivere un programma che legga da tastiera due interi N e K e una sequenza di N stringhe e che stampi le K stringhe più frequenti in essa contenute, in ordine lessicografico.

Si può assumere che:

- non esistano due stringhe con la stessa frequenza;
- il valore di K sia minore o uguale al numero di stringhe distinte;
- le stringhe contengono soltanto caratteri alfanumerici ($a - z$ minuscoli e maiuscoli o numeri, nessuno spazio o punteggiatura) e sono lunghe al più 100 caratteri ciascuna.

Esercizio 1

```
typedef struct {
   char *stringa;
   int freq;
} entry;

int cmpAlfa (const void *p1, const void *p2) {
    return strcmp( *(char **)p1, *(char **)p2 );
}

int cmpEntryAlfa(const void *p1, const void *p2) {
    return strcmp( ((entry *)p1)->stringa, ((entry *)p2)->stringa );
}

int cmpEntryFreq(const void *p1, const void *p2) {
   return ((entry *)p2)->freq - ((entry *)p1)->freq;
}
```

Esercizio 1

```
int main(void) {
 int n, k, i, j; char **stringhe; entry *aggregate;

 stringhe = leggi(&n, &k);
 aggregate = (entry *)malloc(sizeof(entry) * n);
 qsort ( stringhe, n, sizeof(char *), cmpAlfa );
 j = -1;
 for ( i = 0; i < n; i++ ) {
   if ( j >= 0 && !strcmp(aggregate[j].stringa, stringhe[i] ) )
        aggregate[j].freq++;
   else
   {
        j++;
        aggregate[j].stringa = stringhe[i];
        aggregate[j].freq = 1;
   }
}
```

Esercizio 1

```
qsort ( aggregate, j+1, sizeof(entry), cmpEntryFreq );

qsort ( aggregate, k, sizeof(entry), cmpEntryAlfa );

for ( i = 0; i < k; i++ )
  printf("%s\n", aggregate[i].stringa);

  return 0;

}
```

Liste

Liste

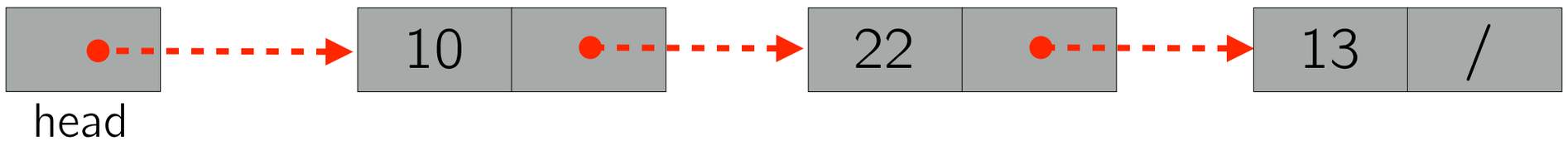
Liste monodirezionali

Liste

Liste monodirezionali

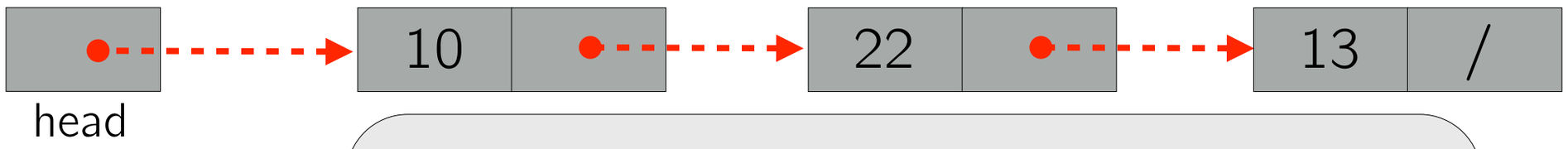
Liste

Liste monodirezionali



Liste

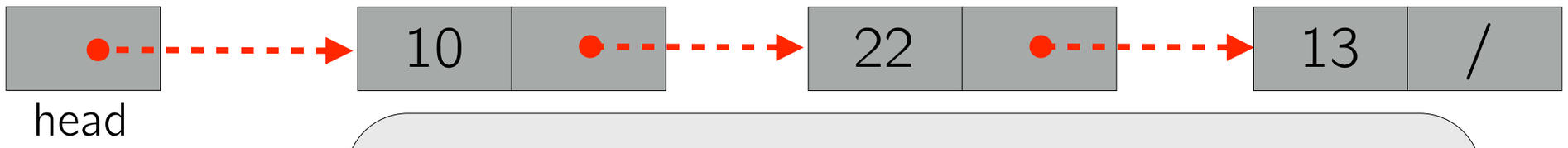
Liste monodirezionali




```
typedef struct _elem {
   int key;
   struct _elem *next;
} elem;
```

Liste

Liste monodirezionali

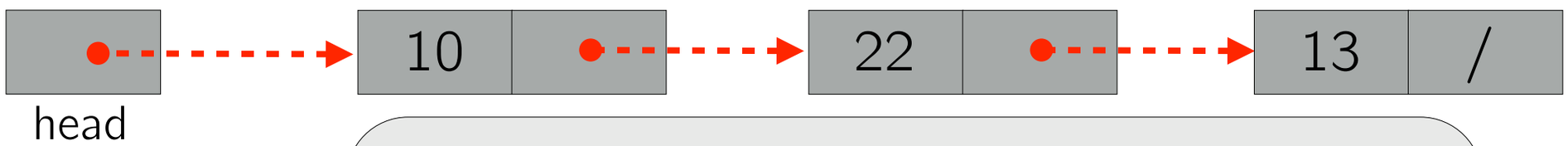



```
typedef struct _elem {
   int key;
   struct _elem *next;
} elem;
```

Liste bidirezionali

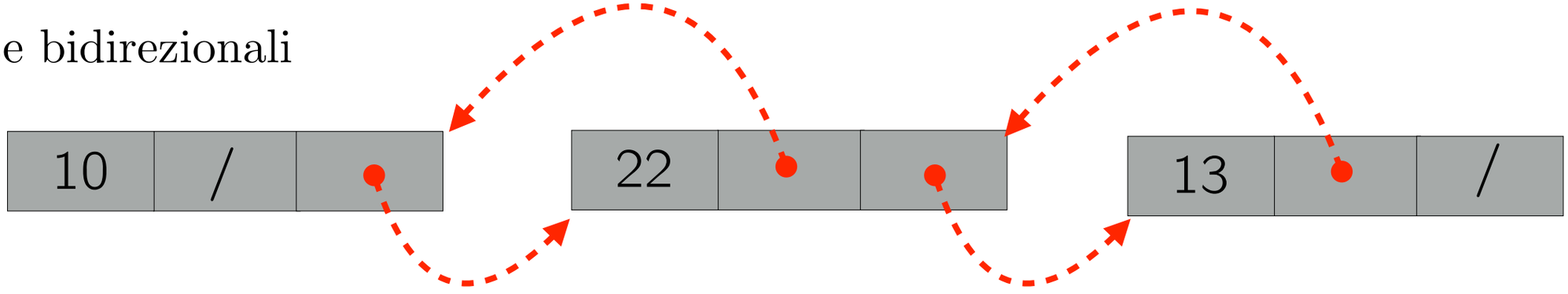
Liste

Liste monodirezionali



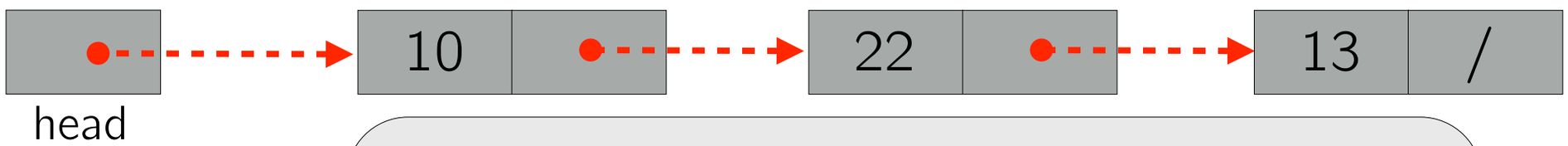

```
typedef struct _elem {
   int key;
   struct _elem *next;
} elem;
```

Liste bidirezionali



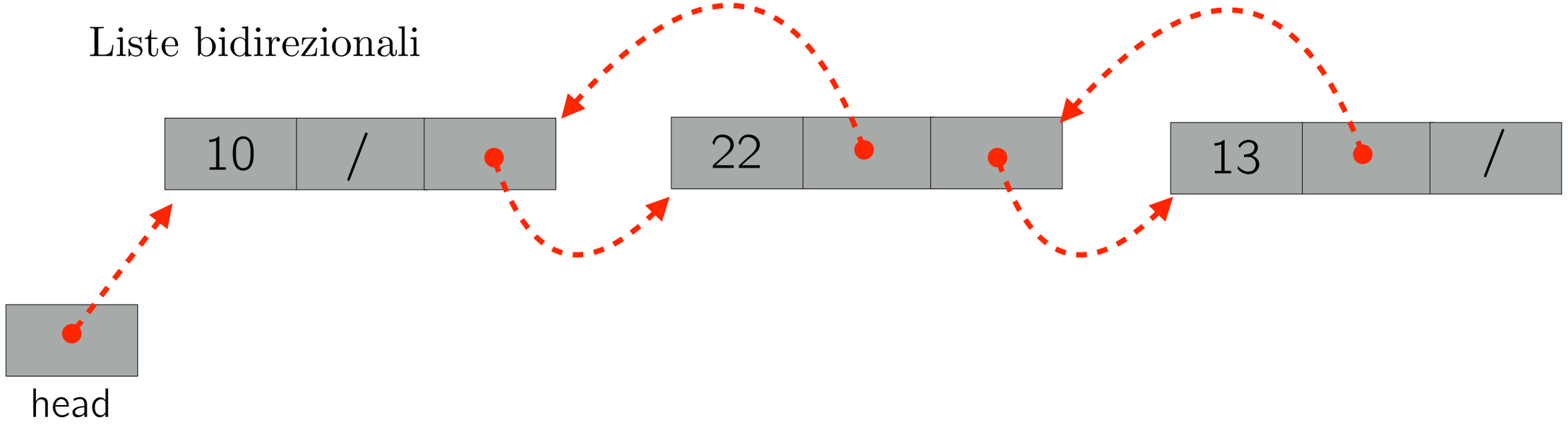
Liste

Liste monodirezionali



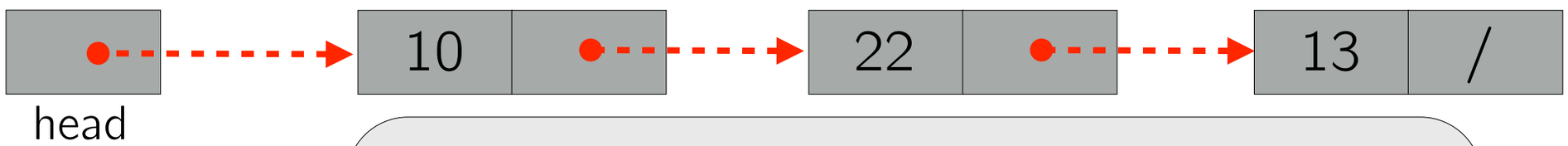

```
typedef struct _elem {
   int key;
   struct _elem *next;
} elem;
```

Liste bidirezionali



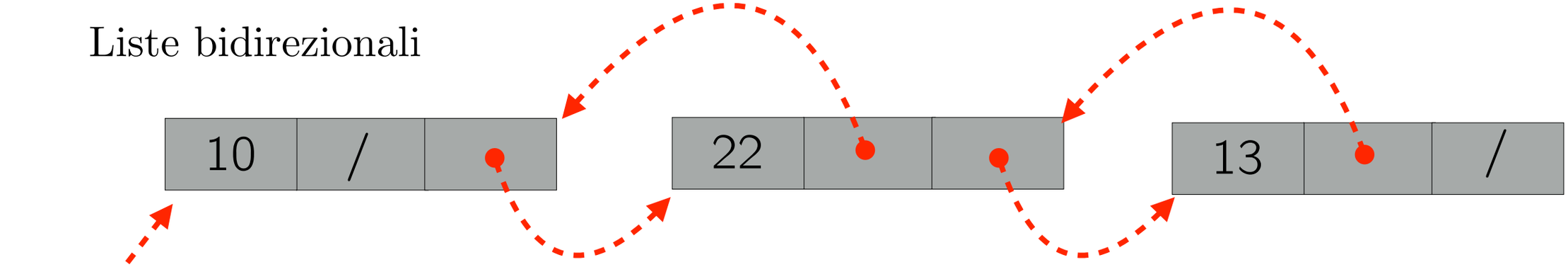
Liste

Liste monodirezionali




```
typedef struct _elem {
   int key;
   struct _elem *next;
} elem;
```

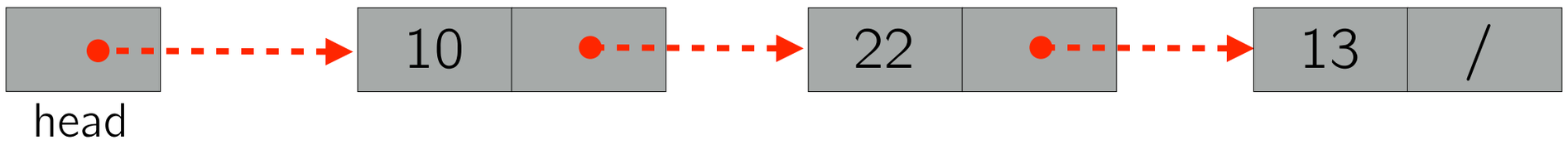
Liste bidirezionali




```
typedef struct _elem {
   int key;
   struct _elem *prev;
   struct _elem *next;
} elem;
```

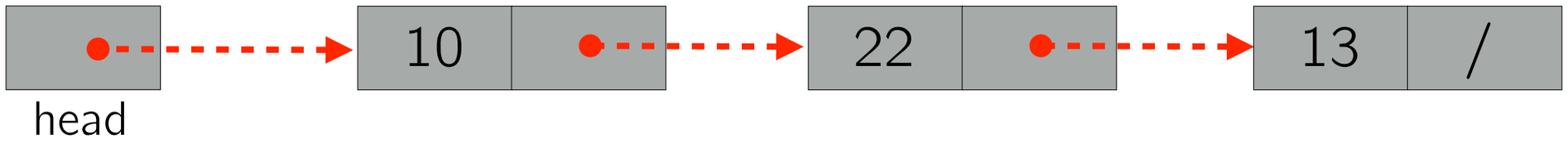
Inserimento in testa

Liste monodirezionali



Inserimento in testa

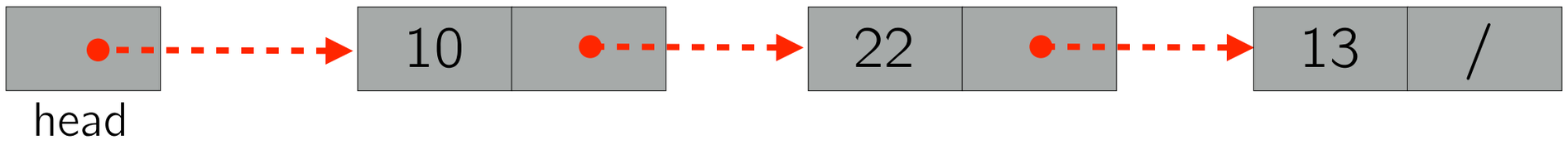
Liste monodirezionali




```
elem* inserisceTesta(elem *head, int key) {
   elem *nuovo = (elem *) malloc(sizeof(elem));
   nuovo->key = key;
```

Inserimento in testa

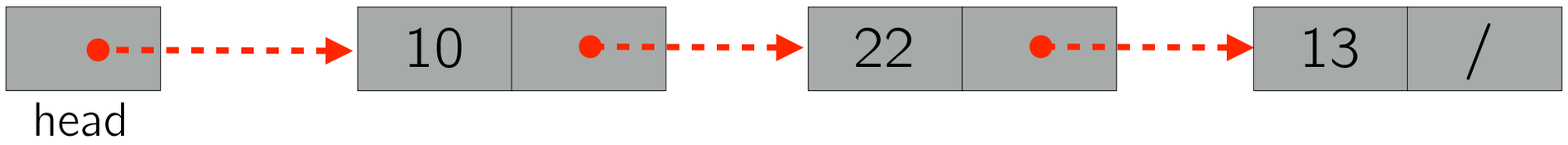
Liste monodirezionali




```
elem* inserisceTesta(elem *head, int key) {
   elem *nuovo = (elem *) malloc(sizeof(elem));
   nuovo->key = key;
```

Inserimento in testa

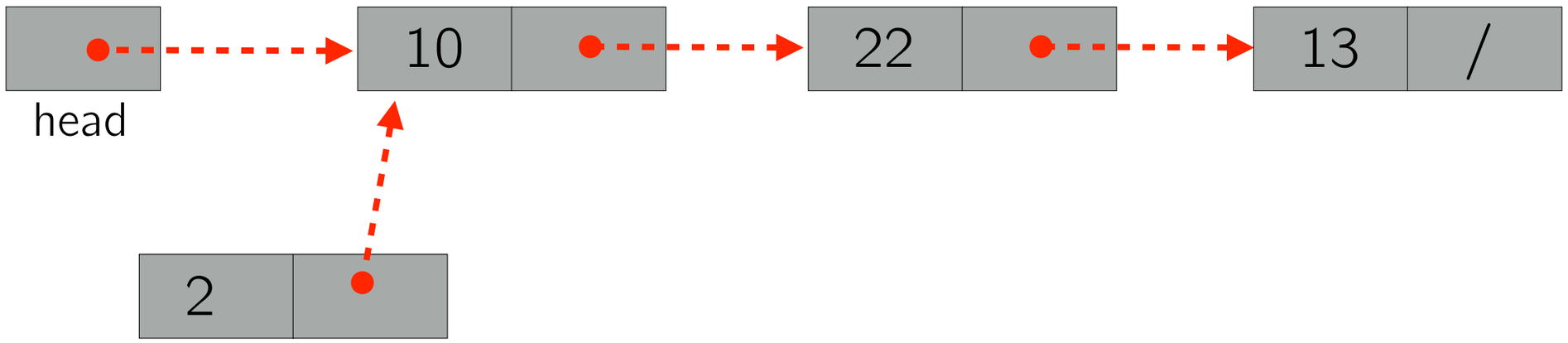
Liste monodirezionali




```
elem* inserisceTesta(elem *head, int key) {
   elem *nuovo = (elem *) malloc(sizeof(elem));
   nuovo->key = key;
   nuovo->next = head; // NULL se la lista è vuota
```

Inserimento in testa

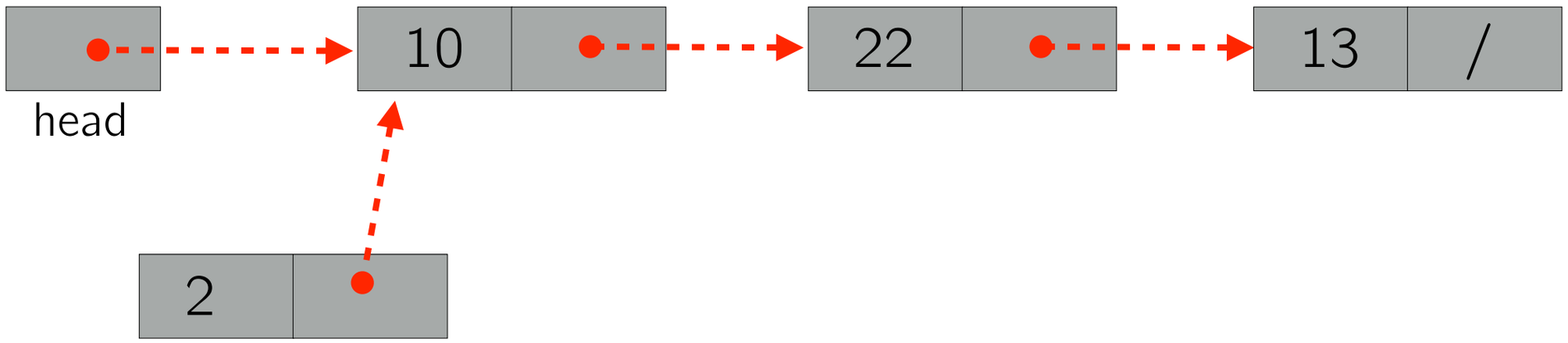
Liste monodirezionali




```
elem* inserisceTesta(elem *head, int key) {
   elem *nuovo = (elem *) malloc(sizeof(elem));
   nuovo->key = key;
   nuovo->next = head; // NULL se la lista è vuota
```

Inserimento in testa

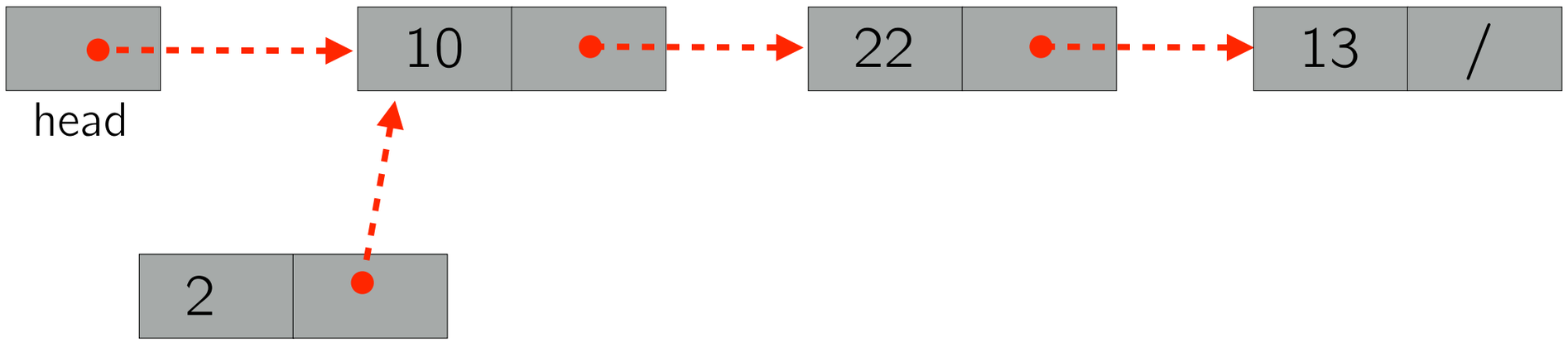
Liste monodirezionali




```
elem* inserisceTesta(elem *head, int key) {
   elem *nuovo = (elem *) malloc(sizeof(elem));
   nuovo->key = key;
   nuovo->next = head; // NULL se la lista è vuota
   return nuovo;
}
```

Inserimento in testa

Liste monodirezionali

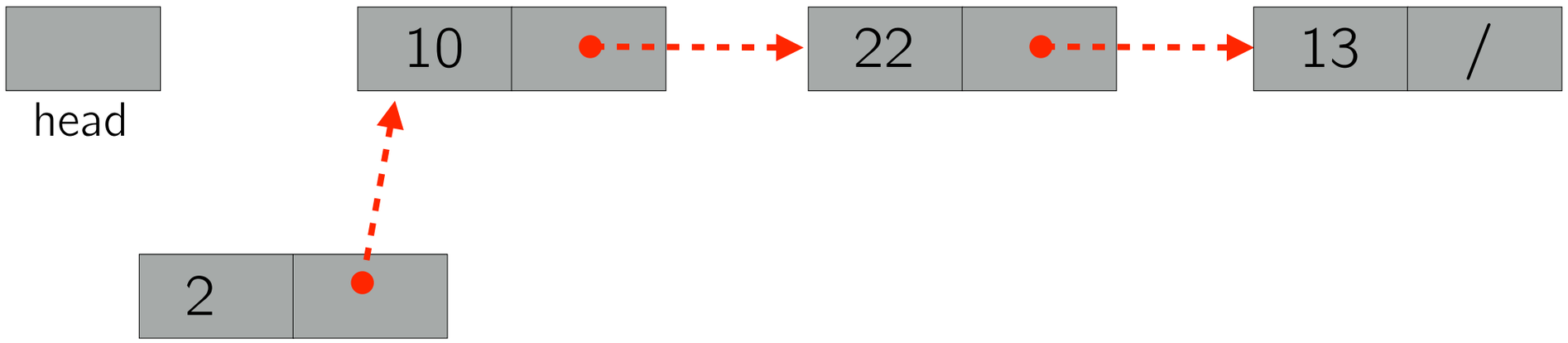



```
elem* inserisceTesta(elem *head, int key) {
   elem *nuovo = (elem *) malloc(sizeof(elem));
   nuovo->key = key;
   nuovo->next = head; // NULL se la lista è vuota
   return nuovo;
}


...
elem* head = NULL;
head = InserisciTesta(head, 2);
```

Inserimento in testa

Liste monodirezionali

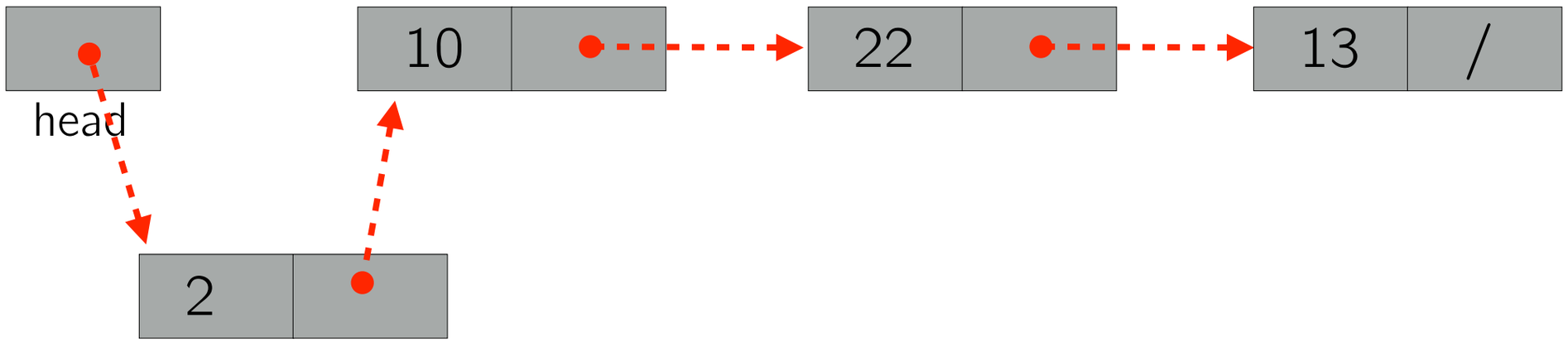



```
elem* inserisceTesta(elem *head, int key) {
   elem *nuovo = (elem *) malloc(sizeof(elem));
   nuovo->key = key;
   nuovo->next = head; // NULL se la lista è vuota
   return nuovo;
}


...
elem* head = NULL;
head = InserisciTesta(head, 2);
```

Inserimento in testa

Liste monodirezionali

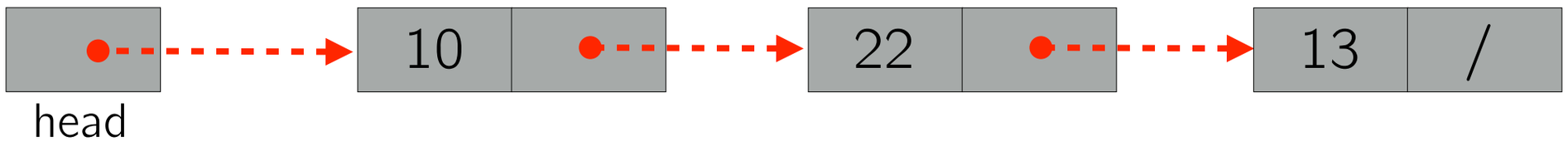



```
elem* inserisceTesta(elem *head, int key) {
   elem *nuovo = (elem *) malloc(sizeof(elem));
   nuovo->key = key;
   nuovo->next = head; // NULL se la lista è vuota
   return nuovo;
}


...
elem* head = NULL;
head = InserisciTesta(head, 2);
```

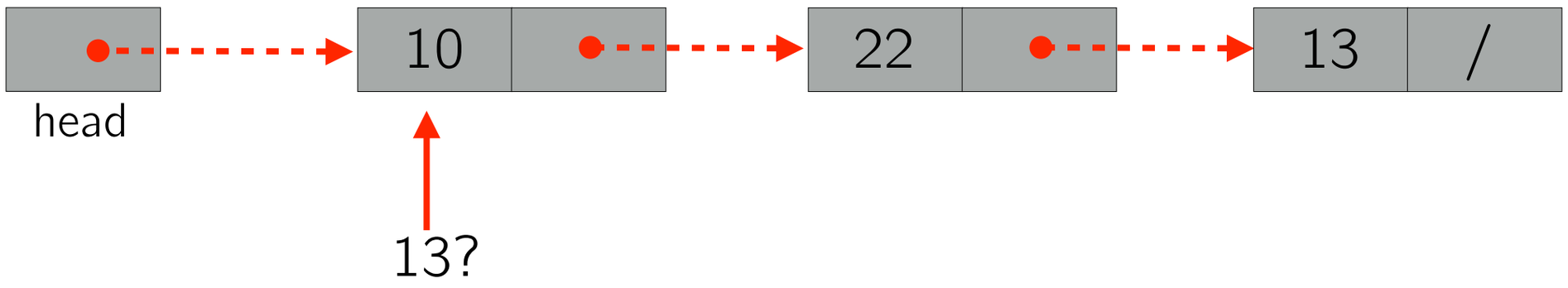
Ricerca di un elemento

Liste monodirezionali



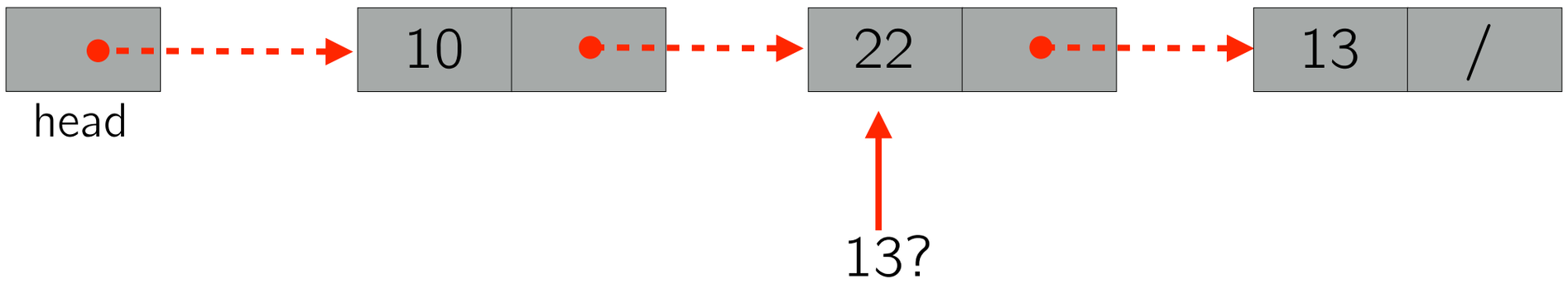
Ricerca di un elemento

Liste monodirezionali



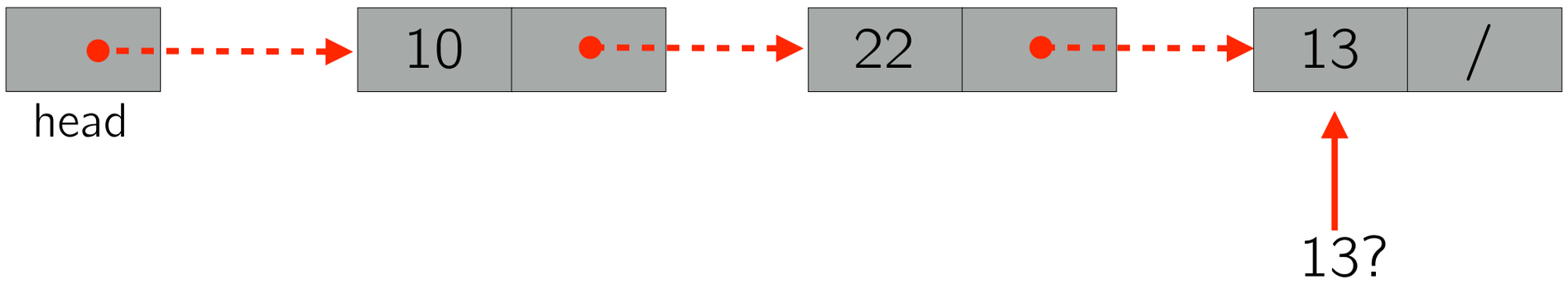
Ricerca di un elemento

Liste monodirezionali



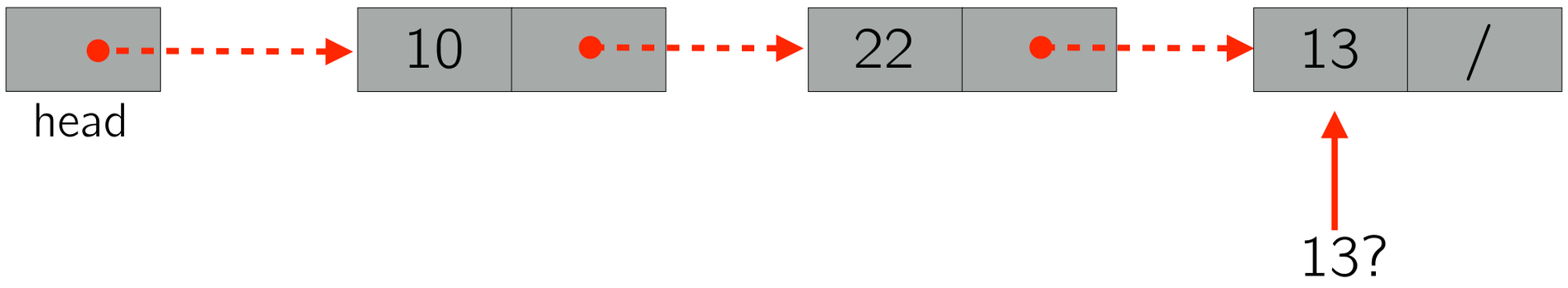
Ricerca di un elemento

Liste monodirezionali



Ricerca di un elemento

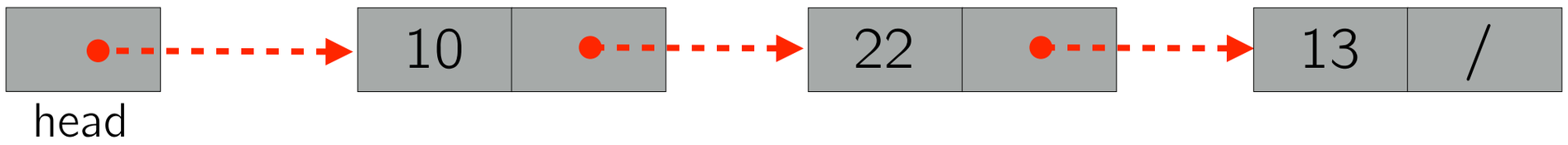
Liste monodirezionali




```
int trovaChiave(elem *head, int key) {
   int i = 0;
   while(head != NULL) {
           if(head->key == key) return i;
           i++;
           head = head->next;
   }
   return -1;
}
```

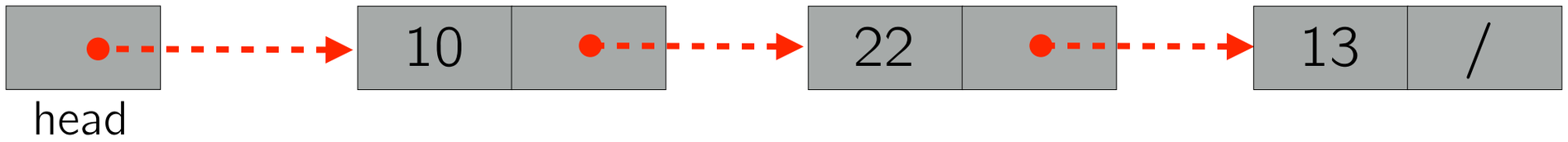
Lunghezza di una lista

Liste monodirezionali



Lunghezza di una lista

Liste monodirezionali




```
int lunghezzaLista(elem *head){
    if(head == NULL) return 0;
    else return 1+lunghezzaLista(head->next);
}
```

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 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 3

Move-to-Front

Scrivere un programma che legga da tastiera una sequenza di n interi **distinti** e li inserisca in una lista monodirezionale (nell'ordine dato). Il programma entra poi in un ciclo nel quale legge un intero i da tastiera e lo cerca nella lista. Se i si trova nella lista stampa la sua posizione (contando da 0) e porta l'elemento in testa alla lista (MTF), altrimenti stampa `-1` e termina.

L'input è formattato come segue:

- la prima riga contiene la lunghezza n (non limitata) della sequenza;
- le successive n righe contengono gli interi che compongono la sequenza, uno per riga;
- segue una sequenza di interi da cercare nella lista (uno per riga), di lunghezza non nota a priori, terminata da un numero non presente nella lista.

L'output contiene:

- le posizioni degli elementi trovati nella lista (si assume che il primo elemento sia in posizione 0), una posizione per riga;
- `-1` se è stato dato in input un numero che non è nella lista, e in tal caso si termina.

Esercizio 4

Lista bidirezionale e conteggio

Scrivere un programma che legga da tastiera una sequenza di n interi **ordinati** e li inserisca in una lista **bidirezionale**. Il programma entra poi in un ciclo nel quale legge un intero i da tastiera e lo cerca nella lista. Se i si trova nella lista, stampa la sua posizione (contando da 0), altrimenti stampa `-1`.

Ogni elemento della lista mantiene anche un contatore che ricorda quante volte è stata cercata la corrispondente chiave. Tutti i contatori sono inizialmente settati a 0. Dopo ogni ricerca si deve garantire che gli elementi della lista siano ordinati in ordine non-crescente di contatore. Se il contatore di un elemento viene incrementato e risulta uguale a quello dell'elemento precedente, i due elementi vanno lasciati nelle loro posizioni.

NOTA: non si devono utilizzare algoritmi di ordinamento, ma osservare che inizialmente la lista è ordinata e che dopo ogni ricerca solo un contatore viene incrementato.

Puzzle

Ciclo in una lista
Interview di Google

Data una lista L, progettare un algoritmo che in tempo $O(n)$ e spazio aggiuntivo costante sia in grado di stabilire se L contiene un ciclo.