Hash Table

La ricerca di un elemento su una lista (semplice o doppia) presenta una complessità di tipo O(n), dato che occorre scandire (al più) l'intera lista per poter trovare l'elemento desiderato.

Le ***hash table*** offrono invece prestazioni più elevate.
Esse sono usate per l'implementazione dei **dizionari**.
Dizionario:
- ***ogni elemento è caratterizzato da un campo "chiave" univoco;***
- ***ricerca e cancellazione avvengono per "chiave".***

Una ***hash table*** è un insieme di *m* liste semplicemente collegate.
Dato un elemento **e** con chiave **k**, la **funzione di hash**:

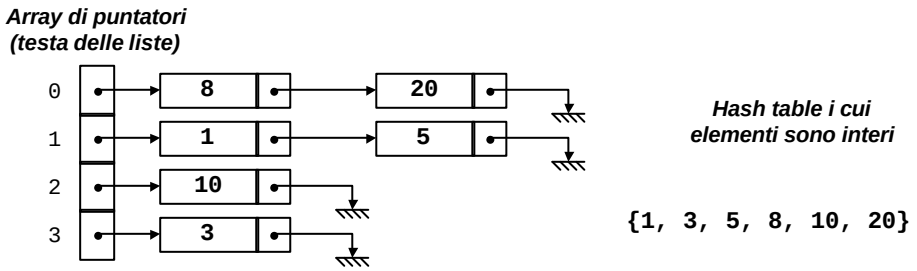
h(k) → lista

stabilisce in quale lista cercare/inserire l'elemento.

La complessità è dunque **O(n/m)**

Hash Table

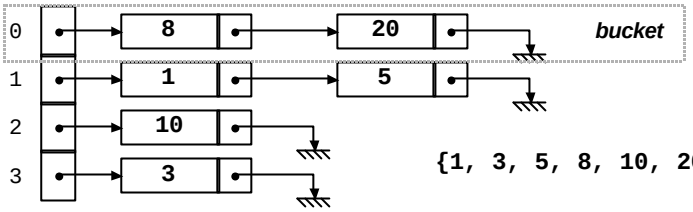
Le rappresentiamo così:




```
h(e) = e mod 4 = e % 4 =  Indice dell'array di
                          puntatori
```

Hash Table

Array di puntatori (testa delle liste)

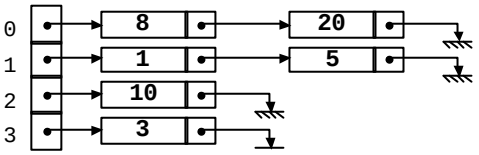



```
#define N 4

typedef struct t_hash_element {
  int                       data;
  struct t_hash_element *   next;
} t_hash_element;

typedef t_hash_element *    t_hash_table [N];

int hash_function (int data)
{
  return data % N;
}
```

3

Hash Table: Ricerca

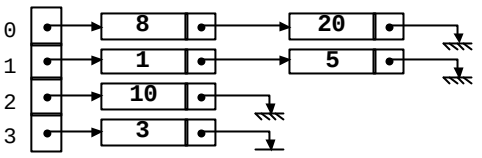
```
#define N 4

typedef struct t_hash_element {
  int                       data;
  struct t_hash_element *   next;
} t_hash_element;

typedef t_hash_element *    t_hash_table [N];

int hash_function (int data)
{
  return data % N;
}

int find_element (t_hash_table table, int value_to_search)
{
  t_list_element * head = table [hash_function (value_to_search)];
  while (head != NULL) {
    if (head->data == value_to_search)
      return 1;
    head = head->next;
  }
  return 0;
}
```



4

Hash Table: Inizializzazione & Inserimento

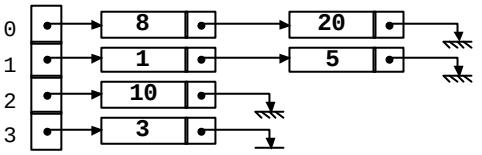
```
void init_table (t_hash_table table)
{
  int i;
  for (i = 0;i < N;i++)
    table [i] = NULL;
}

void add_element (t_hash_table table, int value_to_add)
{
  t_hash_element * newelem;
  int bucket;
  if (find_element (table, value_to_add) == 1)
       return ;  // se l'elemento esiste gia' non lo inserisco
  // alloco l'elemento
  newelem = (t_hash_element *)malloc (sizeof (t_hash_element));
  if (newelem == NULL)
       return;
  newelem->data = value_to_add;
  bucket = hash_function (value_to_add); // determino il bucket
  // inserisco in testa
  newelem->next = table [bucket];
  table [bucket] = newelem;
}
```



5

Hash Table: Visita

```
void dump_table (t_hash_table table)
{
  int i;
  t_hash_element * head;

  for (i = 0;i < N;i++) {

    printf ("Bucket %d:", i);
    head = table [i];
    while (head != NULL) {
      printf ("%d,", head->data);
      head = head->next;
    }
    printf ("\n");
  }
}
```

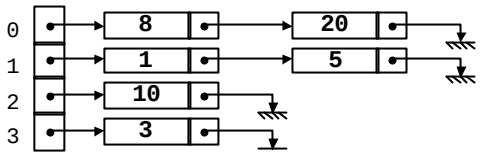


6

Hash Table: Cancellazione elemento

```
void remove_element (t_hash_table table, int value_to_remove)
{
  // determinazione del bucket
  // ricerca dell'elemento nella lista del bucket selezionato
  // eliminazione dell'elemento (stessa modalita' della cancellazione
  // da lista semplicemente collegata)
}
```

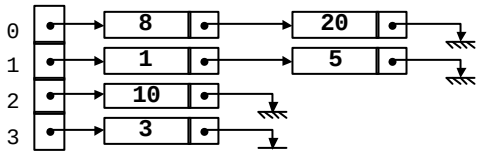
```
#define N 4

typedef struct t_hash_element {
  int                       data;
  struct t_hash_element *   next;
} t_hash_element;

typedef t_hash_element *   t_hash_table [N];

int hash_function (int data)
{
  return data % N;
}
```



7

Hash Table: Cancellazione elemento

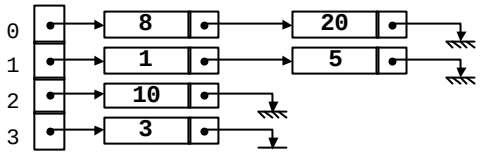
```
void remove_element (t_hash_table table, int value_to_remove)
{
  int bucket;
  t_hash_element * aux, * prev;
  bucket = hash_function (value_to_remove);
  aux = table [bucket];
  prev = NULL;
  while ((aux != NULL) && (aux->data != value_to_remove)) {
        prev = aux;
        aux = aux->next;
  }
  if (aux == NULL)
    return;
  if (prev == NULL) // eliminazione dalla testa
        table [bucket] = aux->next;
  else
        prev->next = aux->next;
  free (aux);
}
```



8

Hash Table: Utilizzo

```
int main(int argc,char *argv[])
{
       t_hash_table mytable;

       init_table (mytable);

       add_element (mytable, 1);
       add_element (mytable, 3);
       add_element (mytable, 8);
       add_element (mytable, 5);
       add_element (mytable, 20);
       dump_table (mytable);

       remove_element (mytable, 20);
       dump_table (mytable);

       remove_element (mytable, 1);
       dump_table (mytable);

       remove_element (mytable, 5);
       dump_table (mytable);
}
```



Hash table con elementi strutturati

```
#define N 4

typedef struct {
  int key;
  char value[10];
} t_element;

typedef struct t_hash_element {
  t_element                data;
  struct t_hash_element *  next;
} t_hash_element;

typedef t_hash_element *   t_hash_table [N];

int hash_function (int key)
{
  return key % N;
}
```

Hash Table: Ricerca

```
typedef struct {
  int key;
  char value[10];
} t_element;

int find_element (t_hash_table table, int key, t_element  * result)
{
  t_list_element * head = table [hash_function (key)];
  while (head != NULL) {
    if (head->data.key == key) {
      *result = head->data;
      return 1;
    }
    head = head->next;
  }
  return 0;
}

...
t_element el_found
if (find_element (mytable, 10, &el_found) == 1)
  printf ("Elemento trovato %d,%s\n", el_found.key, el_found.value);
...
```


Hash Table: Inserimento

```
typedef struct {
  int key;
  char value[10];
} t_element;

void add_element (t_hash_table table, t_element value_to_add)
{
  t_hash_element * newelem;
  t_element el_found;
  int bucket;
  if (find_element (table, value_to_add.key, &el_found) == 1)
       return ;  // se l'elemento esiste gia' non lo inserisco
  // alloco l'elemento
  newelem = (t_hash_element *)malloc (sizeof (t_hash_element));
  if (newelem == NULL)
       return;
  newelem->data = value_to_add;
  bucket = hash_function (value_to_add.key); // determino il bucket
  // inserisco in testa
  newelem->next = table [bucket];
  table [bucket] = newelem;
}
```

Hash Table con chiave alfanumerica

```
typedef struct {
  char key [20];
  char value[10];
} t_element;

// funzione di hash basata sulla lunghezza della chiave
int hash_function (char * s)
{
  return strlen (s) % N;
}

// funzione di hash basata sulla somma dei codici ASCII
int hash_function (char * s)
{
  int i, sum = 0;
  for (i = 0;i < strlen (s);i++)
    sum += s[i];
  return sum % N;
}
```


ESERCIZIO

Considerare un elemento composto da **nome, cognome**, **indirizzo,** e **numero di matricola**.

Gestire un dizionario basato su hash table, in cui la chiave di ricerca e' il numero di **matricola**.

Realizzare le funzioni di:
- ***inserimento***
- ***ricerca***
- ***stampa della tabella***
- ***cancellazione***