

# Realizzazione ADT Dizionario

## Laboratorio di Algoritmi e Strutture Dati

**Domenico Redavid**  
**redavid@di.uniba.it**

**Materiale di base gentilmente concesso  
dal dott. Nicola Di Mauro  
Ricercatore presso l'Univ. di Bari**

# ADT Dizionario

- Un dizionario è una collezione di coppie (*entry*) della forma  $(k,v)$ 
  - $k$  è una chiave
  - $v$  è il valore associato alla chiave  $k$
- Non esistono due entry diverse con la stessa chiave
- Le operazioni permesse potrebbero essere
  - stabilire se il dizionario è vuoto
  - stabilire l'appartenenza di un elemento
  - inserire una *entry*
  - eliminare la *entry* con una specifica chiave
  - Individuare la *entry* con una specifica chiave
  - stabilire la dimensione del dizionario

# Esempio

- Vogliamo costruire una struttura che memorizzi l'insieme delle stringhe  $K = \{\text{ett, tva, tre, fyra, fem, sep, sju, atta, nio, tio, elva, tolv}\}$
- Supponiamo di definire una funzione  $h: K \rightarrow \mathbb{N}$  come riportato nella seguente tabella

| x    | h(x) |
|------|------|
| ett  | 1    |
| tva  | 2    |
| tre  | 3    |
| fyra | 4    |
| fem  | 5    |
| sep  | 6    |
| sju  | 7    |
| atta | 8    |
| nio  | 9    |
| tio  | 10   |
| elva | 11   |
| tolv | 12   |

Possiamo implementare la struttura utilizzando un array di  $n=12$  elementi

- Per l'inserimento, memorizziamo  $x$  in posizione  $h(x)$
- Per cercare  $x$ , verifichiamo se è presente in posizione  $h(x)$

# Hashing

- Vogliamo utilizzare una funzione hash per determinare la posizione di una chiave dell'insieme  $K$
- Se le chiavi sono degli interi a 32 bit allora la cardinalità di  $K$ , ovvero  $|K|$ , è al massimo  $2^{32}$
- Se le chiavi sono delle stringhe allora  $|K|$  è illimitato
- Ci aspettiamo inoltre che il numero delle chiavi da memorizzare nel dizionario sia molto più piccolo di  $|K|$
- Se usiamo un array di  $M$  elementi allora vogliamo definire una funzione hash  $h: K \rightarrow \{0, 1, \dots, M-1\}$
- Collisioni: poiché in generale  $|K| > M$  allora esisteranno più chiavi  $x$  e  $y$ , con  $x \neq y$ , tale che  $h(x) = h(y)$

# Hashing

In generale, è conveniente implementare una funzione hash  $h$  come composizione di due funzioni  $f$  e  $g$

– la funzione  $f$  mappa chiavi in interi

$$f: K \rightarrow \mathbb{N}$$

– la funzione  $g$  mappa interi non negativi in  $\{0, 1, \dots, M-1\}$

$$g: \mathbb{N} \rightarrow \{0, 1, \dots, M-1\}$$

– la funzione  $h$  è poi definita come composizione di  $f$  e  $g$ , ovvero il valore hash di una chiave  $x$  è dato da  $g(f(x))$

Esempio:  $g(f(\text{ciccio})) = g(12) \% 5 = 2$ , supponendo  $M=5$

Dovremmo definire una funzione di hash per ogni tipo (stringa, intero...)

# Esempi

## Definizione di una funzione hash

```
// LIBRERIA DI FUNZIONI HASH
#include <cstdlib>
using namespace std;

typedef unsigned int HashValue;

HashValue hash(string);

// HashValue hash(int);
// ...

// interfaccia

hash.h
```

```
#include <cstdlib>
#include <iostream>
#include "Hash.h"

using namespace std;

HashValue hash(string str)
{
    HashValue hash = 5381;
    int l=str.length();
    int c;
    for (int i=0;i<l;i++)
    {
        int c=str[i]; //il tipo char è già un intero
        hash = hash*33 + c; //funzione hash
    }
    return hash;
}

hash.cpp
```

# ADT Dizionario

## La classe Entry: Interfaccia

```
#ifndef DIZENTRY_H
#define DIZENTRY_H

template<typename Key, typename Value> class Entry {

public:

Entry (const Key&, const Value&); // costruttore
Entry (); // costruttore

// metodi di set

Value getValue(); // restituisce il valore dell'attributo

Key getKey(); // restituisce la chiave

// metodi di set

void setValue(const Value&); // inizializza il valore

void setKey(const Key&); // inizializza la chiave

private:
Key k;
Value v;
};
```

Interfaccia della classe Entry

# ADT Dizionario

## La classe Entry: Realizzazione

```
template<typename Key, typename Value> Entry<Key,Value>::Entry(const Key&  
    chiave, const Value& valore){
```

```
    k=chiave;  
    v=valore;  
}
```

```
template<typename Key, typename Value> Entry<Key,Value>::Entry(){ }
```

```
template<typename Key, typename Value> Key Entry<Key,Value>::getKey(){  
    return k;  
}
```

```
template<typename Key, typename Value> Value Entry<Key,Value>::getValue(){  
    return v;  
}
```

```
template<typename Key, typename Value> void Entry<Key,Value>::setKey(const  
    Key& chiave){  
    k=chiave;  
}
```

```
template<typename Key, typename Value> void Entry<Key,Value>::setValue(const  
    Value& valore){  
    v=valore;  
}
```

Realizzazione della classe Entry



# ADT Dizionario

## Realizzazione con liste di trabocco

```
#ifndef DIZ_H
#define DIZ_H
#include <iostream>
#include "Hash.h"
#include "listap.h"
#include "Entry.h"

template<typename K, typename E> class Dizionario {
// E=tipo elemento, K=tipo chiave
typedef Entry<K,E> entry;

public:
Dizionario(unsigned int); // costruttore
~Dizionario(); // distruttore
bool vuoto(); // restituisce true se il dizionario è vuoto

bool appartiene(const K& ); // restituisce true se l'elemento appartiene al
    dizionario

E recupera(const K&); // restituisce l'elemento corrispondente alla chiave (se
    esiste)

void inserisci(const K&, const E&); // inserisce un elemento nel dizionario
void cancella(const K&); // elimina l'elemento
unsigned int dimensione() ; // restituisce il numero elementi del dizionario

void stampa();
```

Interfaccia della classe Dizionario

# ADT Dizionario

## Realizzazione con liste di trabocco

```
private:  
unsigned int H(const K&); // calcola il valore hash dell'elemento object  
  
unsigned int lunghezza; // lunghezza massima del dizionario  
  
unsigned int nelementi; // elementi presenti in un certo istante nel dizionario  
  
circLista< Entry <K,E> >* table; // liste di trabocco per le entry  
  
};
```

Parte privata della classe Dizionario

# ADT Dizionario

## Realizzazione metodi

```
// costruttore
template <typename K, typename E> Dizionario<K,E>::Dizionario(unsigned int n){
lunghezza = n;
nelementi = 0;
table = new cirLista< entry >[lunghezza]; // vettore di liste circolari
}
// distruttore
template <typename K, typename E> Dizionario<K,E>::~~Dizionario(){
delete[] table; // cancella il vettore table
}
// dizionario vuoto
template<typename K, typename E> bool Dizionario<K,E>::vuoto(){
return (nelementi == 0);
}
// calcola il valore hash dell'elemento
template<typename K, typename E> unsigned int Dizionario<K,E>::H(const K& chiave)
{
return (hash(chiave) % lunghezza); // funzione hash modulo lunghezza
}
// numero elementi del dizionario
template<typename K, typename E> unsigned int Dizionario<K,E>::dimensione(){
return nelementi;
}
// stampa elementi
template<typename K, typename E> void Dizionario<K,E>::stampa(){
for (int i=0;i<lunghezza;i++)
{
table[i].stampaLista(); // stampa la lista circolare presente nella posizione i
}
}
}
```

# ADT Dizionario

## Realizzazione operatori appartiene e recupera

```
// appartiene al dizionario
template<typename K, typename E> bool Dizionario<K,E>::appartiene(const K& key){
    Cella<entry>* iter; // puntatore a cella della lista
    bool trovato = false;
    unsigned int pos = H(key); // ottieni la posizione nel vettore mediante key
    iter = table[pos].primoLista(); // setta iter al primo elemento della lista
    // controlliamo se la chiave è contenuta in una delle entry delle celle della lista
    while (!table[pos].finelista(iter) && !trovato){
        if (key == table[pos].leggilista(iter).getKey())
            trovato = true;
        iter = table[pos].succlista(iter);
    }
    return(trovato);
}

// recupera da dizionario
template<typename K, typename E> E Dizionario<K,E>::recupera(const K& key){
    Cella <entry>* iter; // puntatore a cella della lista
    bool trovato = false;
    E elem; // elemento da restituire
    unsigned int pos = H(key); // ottieni la posizione nel vettore mediante key
    iter = table[pos].primoLista(); // setta iter al primo elemento della lista
    while (!table[pos].finelista(iter) && !trovato){
        if (key == table[pos].leggilista(iter).getKey()) {
            trovato = true;
            elem=table[pos].leggilista(iter).getValue(); // elemento associato alla chiave
        }
        iter = table[pos].succlista(iter);
    }
    return(elem);
}
```

# ADT Dizionario

## Realizzazione operatori inserimento e cancell.

```
// inserisci nel dizionario
template<typename K, typename E> void Dizionario<K,E>::inserisci(const K& chiave,
    const E& element){
    entry e(chiave,element); // creo una entry
    Cella <entry>* posizione=table[H(chiave)].primoLista(); // prendo la prima posizione
    table[H(chiave)].inslista(e,posizione); // e inserisco la entry
    nelementi++;
}

//cancella dal dizionario
template<typename K, typename E> void Dizionario<K,E>::cancella(const K& chiave){
    Cella <entry>* iter;
    bool trovato = false;
    unsigned int pos = H(chiave);
    iter = table[pos].primoLista();
    // cerco nella lista se c'è una entry con quella chiave
    while (!table[pos].finelista(iter) && !trovato){
        if (chiave == table[pos].leggilista(iter).getKey())
            trovato = true;
        else
            iter = table[pos].succlista(iter);
    }
    // se la trovo, cancello la cella
    if (trovato){
        table[pos].canclista(iter);
        nelementi--;
    }
}
```

# Test

## Dizionario di Studenti

```
#ifndef STUDENTE_H
#define STUDENTE_H

#include <string>
#include <iostream>
using namespace std;

class Studente {
friend ostream &operator<<(ostream&, const Studente &);
public:
    Studente(string, string, string);
    Studente();
    ~Studente();
    string getNome() const;
    string getCognome() const ;
    string getMatricola() const;
    void stampa() const;
private:
    string nome;
    string cognome;
    string matricola;
};

#endif
```

```
#include <iostream>
#include "Studente.h"
using namespace std;

Studente::Studente(string n, string c, string m)
{   nome=n;
    cognome=c;
    matricola = m; }
Studente::Studente()
{   nome="";
    cognome="";
    matricola = ""; }
Studente::~~Studente() {}
string Studente::getNome() const
{   return nome;}

string Studente::getCognome() const
{   return cognome;
}

string Studente::getMatricola() const
{   return matricola;
}

void Studente::stampa() const
{
    cout << "Nome: " << nome << "\n";
    cout << "Cognome: " << cognome << "\n";
    cout << "Matricola: " << matricola << "\n";
}

ostream &operator<<(ostream &os, const Studente &s)
{
    return (os << s.getMatricola() << " - " <<
s.getCognome() << " "<<s.getNome());
}
```

# Test

## Dizionario di Studenti: Main

```
#include <cstdlib>
#include <iostream>
#include "Dizionario.h"
#include "Studente.h"
using namespace std;

int main(int argc, char *argv[])
{
    Studente s1("paolo","rossi","12354");
    Studente s2("mario","bianchi","12366");
    Studente s3("antonio","verdi","12386");
    Dizionario<string,Studente> d(4);
    // proviamo gli operatori
    d.inserisci("12354", s1);
    cout << d.appartiene("12354");
    cout << d.appartiene("12366");
    cout << d.appartiene("12386");
    cout << "Dovrebbe aver stampato 100\n";
    cout << "Dimensione del dizionario: " <<
d.dimensione() << "\n";
    d.inserisci("12366", s2);
    d.inserisci("12386", s3);
    cout << "ho fatto 3 inserimenti\n";
    cout << "Dimensione del dizionario: " <<
d.dimensione() << "\n";
    cout << d.appartiene("12354");
    cout << d.appartiene("12366");
    cout << d.appartiene("12386");
    cout << "Dovrebbe aver stampato 111\n";
    d.cancella("12386");
    d.cancella("12354");
    cout << "ho fatto 2 cancellazioni\n";
    cout << "Dimensione del dizionario: " <<
d.dimensione() << "\n";
```

```
    cout << d.appartiene("12354");
    cout << d.appartiene("12366");
    cout << d.appartiene("12386");
    cout << d.appartiene("283808");
    cout << "Dovrebbe aver stampato 0100\n";
    d.recupera("12366").stampa();
    d.recupera("12386").stampa();
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

# Esercizio

- La frequenza di un corso universitario da parte di una classe di studenti è registrata su un foglio presenze in cui sono annotati, per ogni lezione, nome, cognome e numero di matricola di ogni studente. Siano dati i fogli presenza di due lezioni.
- A) Costruire una funzione che, data una matricola, indichi se lo studente era presente ad almeno una delle lezioni, stampandone i dati (Aiuto: usare la struttura dati lista per rappresentare le presenze giornaliere)
- B) Costruire una funzione che conti il numero degli studenti presenti alla prima ma non alla seconda lezione



# Svolgimento punto A)

```
void presente (string matricola, cirLista<Studente>& l1, cirLista<Studente>& l2)
{
    Dizionario<string,Studente> d1(3);
    Dizionario<string,Studente> d2(3);
    cirLista<Studente>::posizione indiceElemento = l1.primoLista();
    while (!l1.finelista(indiceElemento))
    {
        d1.inserisci(l1.leggilista(indiceElemento).getMatricola(),l1.leggilista(indiceElemento));
        indiceElemento=l1.succlista(indiceElemento);
    }
    if (d1.appartiene(matricola))
    {
        cout << "Studente " << matricola << " presente\n";
        d1.recupera(matricola).stampa();
    }
    else
    {
        indiceElemento = l2.primoLista();
        while (!l2.finelista(indiceElemento))
        {
            d2.inserisci(l2.leggilista(indiceElemento).getMatricola(),l2.leggilista(indiceElemento));
            indiceElemento=l2.succlista(indiceElemento);
        }
        if (d2.appartiene(matricola))
        {
            cout << "Studente " << matricola << " presente\n";
            d2.recupera(matricola).stampa();
        }
        else
            cout << "Studente " << matricola << " assente\n";
    }
}
```

# Test

```
int main(int argc, char *argv[])
{
    circLista<Studiante> lezione1;
    circLista<Studiante> lezione2;
    // studenti
    Studiante s1("paolo","rossi","12354"); // lez1
    Studiante s2("mario","bianchi","12366"); // lez1
    Studiante s3("antonio","verdi","12386"); // lez1 + lez2
    Studiante s4("pinco","pallino","12554"); // lez2
    Studiante s5("rosa","berardi","14386"); // lez1
    Studiante s6("arianna","longo","12777"); // lez 2
    Studiante s7("anna","marchesi","13732"); // lez 2
    Studiante s8("giulio","bernardi","15690"); // lez1 + lez 2
    // riempiamo la prima lista
    circLista<Studiante>::posizione indiceElemento = lezione1.primoLista();
    lezione1.inslista(s1,indiceElemento=lezione1.succlista(indiceElemento));
    lezione1.inslista(s2,indiceElemento=lezione1.succlista(indiceElemento));
    lezione1.inslista(s3,indiceElemento=lezione1.succlista(indiceElemento));
    lezione1.inslista(s5,indiceElemento=lezione1.succlista(indiceElemento));
    lezione1.inslista(s8,indiceElemento=lezione1.succlista(indiceElemento));
    // riempiamo la seconda lista
    indiceElemento = lezione2.primoLista();
    lezione2.inslista(s3,indiceElemento=lezione2.succlista(indiceElemento));
    lezione2.inslista(s4,indiceElemento=lezione2.succlista(indiceElemento));
    lezione2.inslista(s6,indiceElemento=lezione2.succlista(indiceElemento));
    lezione2.inslista(s7,indiceElemento=lezione2.succlista(indiceElemento));
    lezione2.inslista(s8,indiceElemento=lezione2.succlista(indiceElemento));
    // stampo i dati acquisiti
    lezione1.stampaLista();
    lezione2.stampaLista();
    presente("13732", lezione1,lezione2);
    presente("13731", lezione1,lezione2);
    presente("12386", lezione1,lezione2);
}
```

# Svolgimento punto B)

```
int contaAnnoati (cirLista<Studente>& l1, cirLista<Studente>& l2)
{
    Dizionario<string,Studente> d(3);
    cirLista<Studente>::posizione indiceElemento = l2.primoLista();
    while (!l2.finelista(indiceElemento))
    {
        d.inserisci(l2.leggilista(indiceElemento).getMatricola(),l2.leggilista(indiceElemento));
        indiceElemento=l2.succlista(indiceElemento);
    }
    // ho riempito il dizionario
    int counter=0;
    indiceElemento = l1.primoLista();
    while (!l1.finelista(indiceElemento))
    {
        if (!d.appartiene(l1.leggilista(indiceElemento).getMatricola()))
            counter++;
        indiceElemento=l1.succlista(indiceElemento);
    }
    return counter;
}
```

# Esercizio

- Si supponga di disporre delle rilevazioni delle temperature minime e massime giornaliere di alcune città europee
- Si realizzi una funzione che, data una città, restituisca le due temperature registrate.