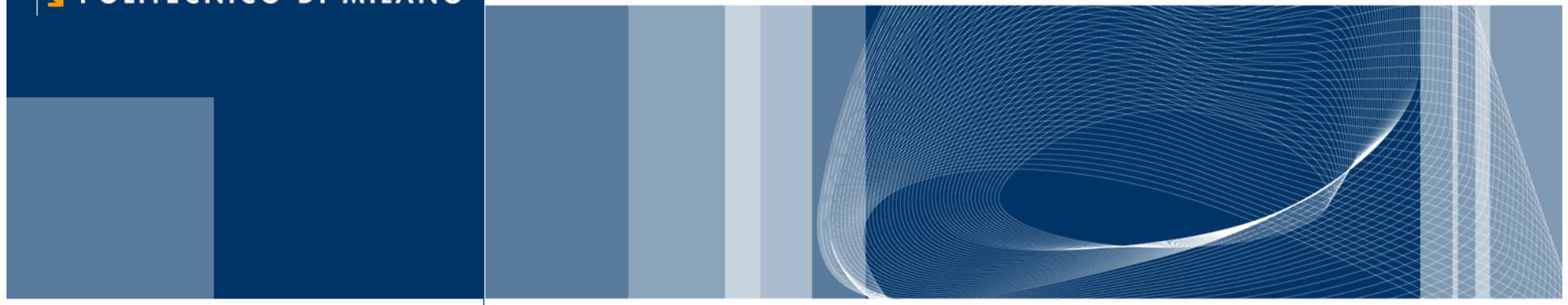
POLITECNICO DI MILANO

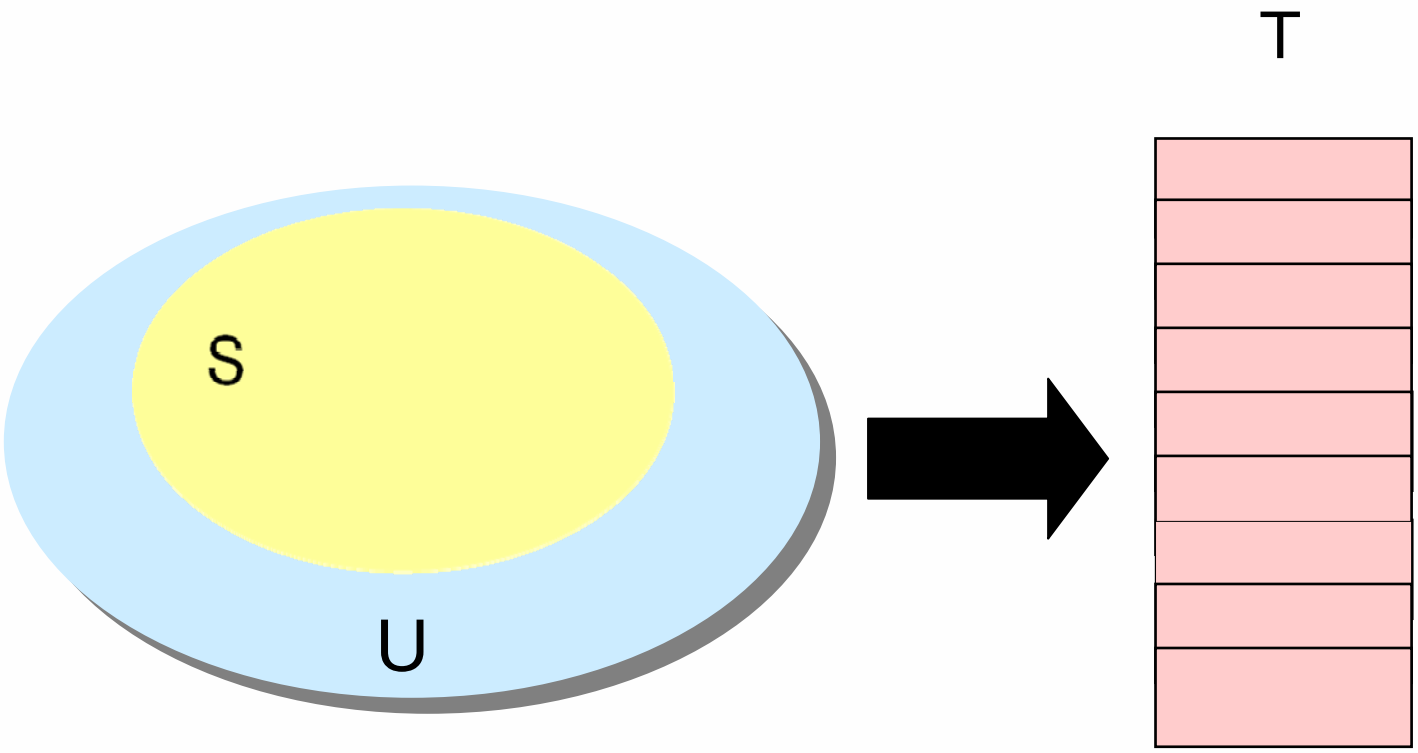


# Algoritmi e Strutture Dati

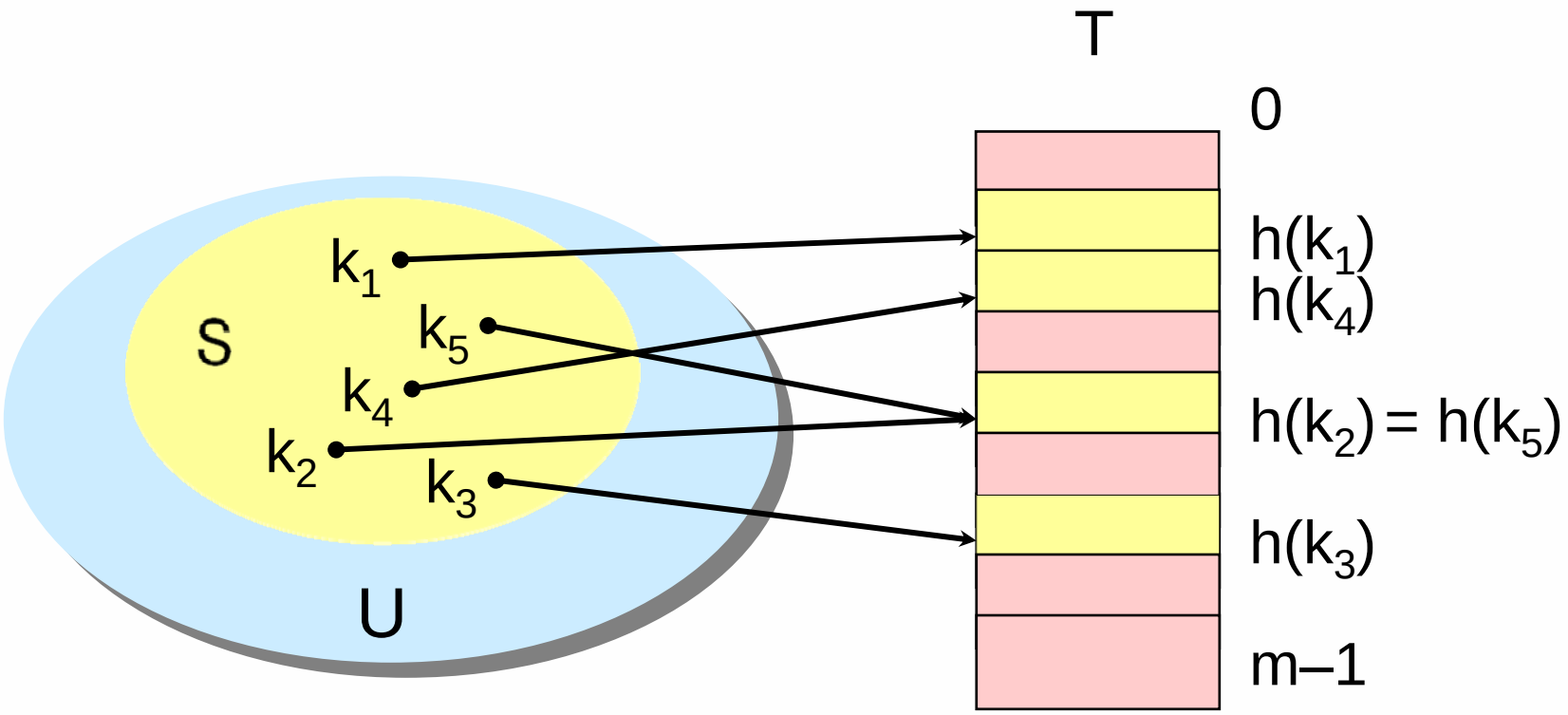
## Laboratorio 01/12/2008

Daniele Loiacono

# Hashing



# Hashing

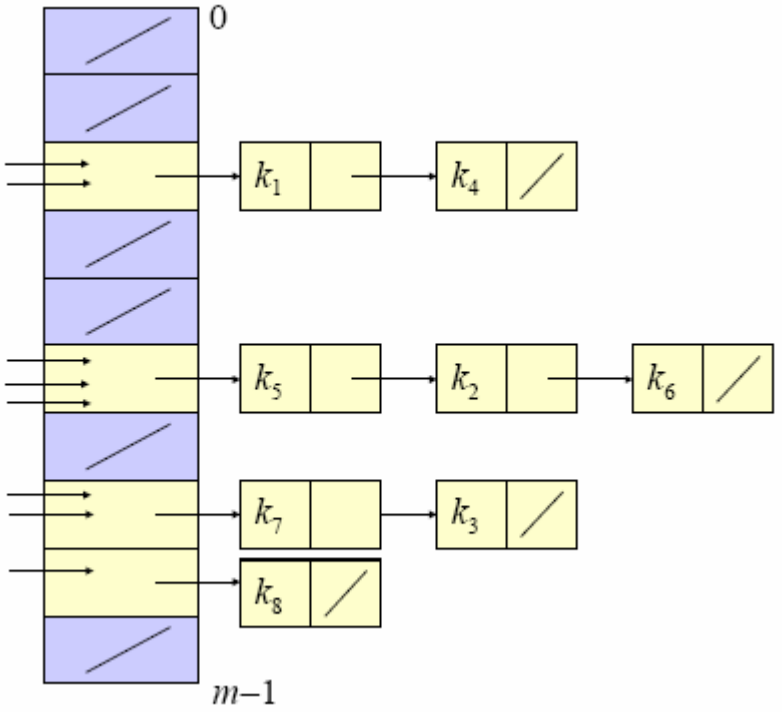



**Problema**

Quando una chiave $k_i$ viene mappata in uno slot già occupato si genera una collisione

# Risoluzione delle collisioni: Concatenamento (chaining)

- Gli elementi con lo stesso valore hash h vengono memorizzati in una linked list
- Si memorizza un puntatore alla testa della lista nello slot A[h] della tabella hash
- Operazioni
  - Insert: inserimento in testa
  - Search, Delete: richiedono di scandire la lista alla ricerca della chiave



# Esercizio 1

- Scrivere il template di una classe che implementi una tabella di hashing. Usare la seguente classe come base:

```
template <class Elem> class TabellaHashBase {
public:
   virtual bool insert(Elem e, int key) = 0;
   virtual bool search(Elem &e, int key) = 0;
   virtual bool del(int key) = 0;
};
```

# Probing

- Ciascun elemento *i* ha una home position $\boldsymbol{h}(k_i)$.
- Se un altro elemento occupa la home position di *i* occorre cercare un altro slot per memorizzare *i*.
- La politica usata per individuare uno slot libero è detta politica di risoluzione delle collisioni.
- La stessa politica deve essere seguita anche per cercare elementi nella tabella.
- Probe sequence: La serie di posizioni della tabella "visitate" seguendo la politica di risoluzione delle collisioni.

# Algoritmo di Inserimento



```
Hash-Insert(A, k)
1. i := 0
2. repeat j := h(k) + p(k, i)
3.     if A[j] = nil then
4.             A[j] := k
5.             return j
6.     else
7.             i := i + 1
8. until i = m
9. error “hash table overflow”
```

# Algoritmo di Ricerca



```
Hash-Search (A, k)
1. i := 0
2. repeat j := h(k) + p(k, i)
3.     if A[j] = k then
4.             return j
5.     i := i + 1
6. until A[j] = nil or i = m
7. return nil
```

# Linear Probing

```
p(k, i) = i;
```

- Cerca un posto nello slot successivo.
- La tabella viene gestita in maniera circolare.
- Per evitare di entrare in un loop infinito, almeno un elemento della tabella deve sempre essere vuoto

# Improved Linear Probing

```
p(k, i) = c*i;
```

- M e c devono essere scelti con attenzione
  - La probe sequence deve consentire di visitare tutte le posizioni della tabella
  - Occorre scegliere un valore di c primo rispetto al valore di M.

# Quadratic Probing

$$h(k, i) = i^2;$$

- Esempio:
  - M=101
  - h(k1)=30, h(k2) = 29.
  - Probe sequence di k1:  30, 31, 34, 39.
  - Probe sequence di k2:  29, 30, 33, 38.

# Double Hashing

```
p(k, i) = i * h2(k)
```

- h2 è la funzione di hashing secondaria

- Esempio:
  - M=101
  - h(k1)=30, h(k2)=28, h(k3)=30.
  - h2(k1)=2, h2(k2)=5, h2(k3)=5.
  - Probe sequence di k1: 30, 32, 34, 36.
  - Probe sequence di k2: 28, 33, 38,43.
  - Probe sequence di k3: 30, 35, 40, 45.

# Esercizio 2

- Scrivere il template di una classe che implementa un elemento in una tabella di hashing. Usare la seguente classe come linea guida:

```
template <class Elem> class HashElem
{
public:
        HashElem();
        HashElem(Elem e, int key);
        void setNil(bool nil);
        void setTomb(bool tomb);
        void setKey(int key);
        void setElem(Elem e);
        bool getNil();
        bool getTomb();
        int getKey();
        Elem getElem();
        bool isFree();
};
```

# Esercizio 3

- Implementare l'hash function con il metodo della divisione usando la seguente classe base:

```
class HashFunctionBase
{
public:
   virtual int h(int key) = 0;
};
```

- Ipotizzare che la dimensione della tabella di hashing sia un parametro del costruttore

# Esercizio 4

- Implementare le probe function descritte in precedenza usando la seguente classe base:

```
class ProbeFunctionBase
{
public:
   virtual int p(int key, int i) = 0;
};
```

# Esercizio 5

- Scrivere il template di una classe che implementi una tabella di hashing con risoluzione delle collisioni. Usare la seguente classe come base:

```
template <class Elem> class TabellaHashBase {
public:
   virtual bool insert(Elem e, int key) = 0;
   virtual bool search(Elem &e, int key) = 0;
   virtual bool del(int key) = 0;
};
```

- Ipotizzare che siano parametri del costruttore:
  - la dimensione della tabella
  - la hasfunction
  - la probe function