# Tabelle hash

Strutture efficienti per implementare un dizionario quando non è ragionevole avere un array con tante posizioni quante sono le possibili chiavi (*direct addressing*).

Si utilizza una *funzione hash* per calcolare l'indice in cui conservare l'elemento con chiave $k$:

$$h : U \to \{0, 1, \ldots, m-1\},$$

dove $U$ è l'universo delle chiavi e $m$ è la dimensione della tabella nella quale vengono inseriti gli oggetti. È utile definire il *load factor* $\lambda = \frac{n}{m}$, dove $n$ è il numero di elementi presenti all'interno della tabella.

## Collisioni

Tipicamente $|U| \gg m$, quindi è necessario gestire le collissioni (quando due chiavi hanno lo stesso hash).

Un possibile approccio è il *chaining*, ovvero inserire gli elementi che hanno lo stesso hash in una lista collegata: in questo modo l'inserimento (se non si controlla se l'elemento è già presente) e la rimozione (se la lista è doppiamente collegata) sono O(1).

Alternativamente, il metodo dell'*open addressing* prevede di trovare uno slot libero in base ad una funzione di *probing*. In questo caso la tabella si può riempire completamente e vale sempre $\lambda \leq 1$. La ricerca avviene controllando tutti gli slot della sequenza di probing finché non si incontra quello desiderato, uno slot vuoto o la prima posizione per la seconda volta. Si può ottimizzare la cancellazione sostituendo l'elemento da eliminare con un valore considerato nullo dalla procedura di inserimento ma occupato da quella di ricerca.