بسم الله الرحمن الرحیم

ساختمان‌های داده

جلسه ۲۱

مجتبی خلیلی
دانشکده برق و کامپیوتر
دانشگاه صنعتی اصفهان

درخت

- B-Tree و Splay tree

درخت

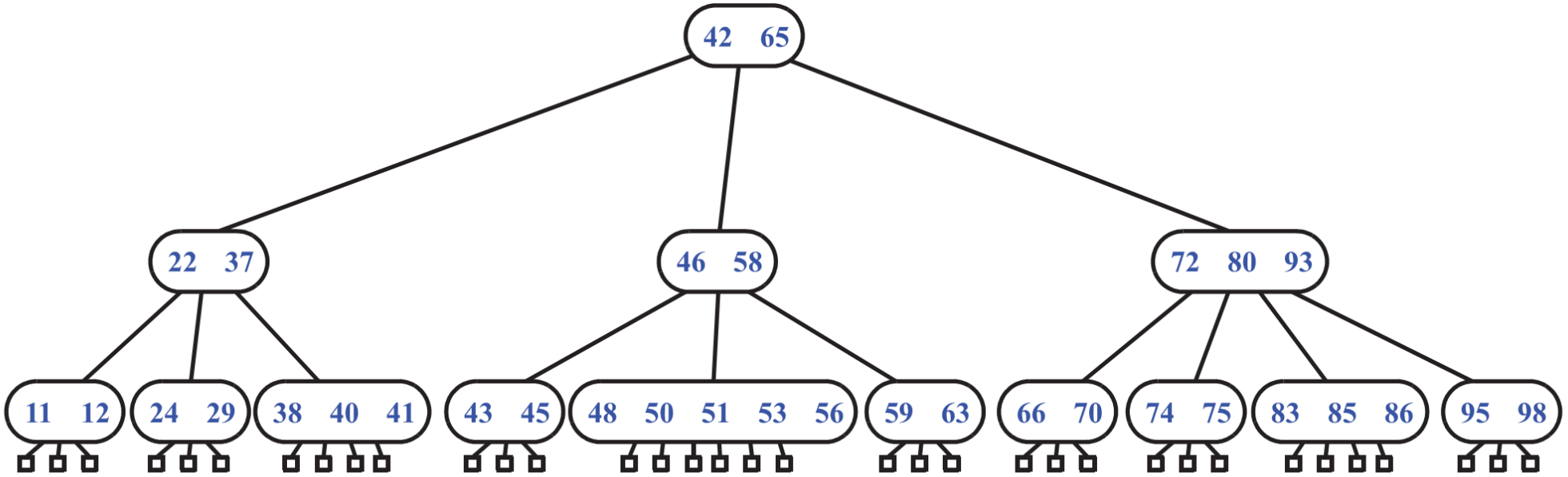


Figure 14.6: A B-tree of order 6.

C++ STL

STL Container	*Description*
vector	Vector
deque	Double ended queue
list	List
stack	Last-in, first-out stack
queue	First-in, first-out queue
priority_queue	Priority queue
set (and multiset)	Set (and multiset)
map (and multimap)	Map (and multi-key map)

RB-tree

- دیگران

11.4.1 The Set ADT

A ***set*** is a collection of distinct objects. That is, there are no duplicate elements in a set, and there is no explicit notion of keys or even an order. Even so, if the elements in a set are comparable, then we can maintain sets to be ordered. The fundamental functions of the set ADT for a set S are the following:

insert(e): Insert the element e into S and return an iterator referring to its location; if the element already exists the operation is ignored.

find(e): If S contains e, return an iterator p referring to this entry, else return end.

erase(e): Remove the element e from S.

begin(): Return an iterator to the beginning of S.

end(): Return an iterator to an imaginary position just beyond the end of S.

C++ STL

std: :map<int, int> Mymap;

size():	Return the number of elements in the map.
empty():	Return true if the map is empty and false otherwise.
find(k):	Find the entry with key k and return an iterator to it; if no such key exists return end.
operator[k]:	Produce a reference to the value of key k; if no such key exists, create a new entry for key k.
insert(pair(k, v)):	Insert pair (k, v), returning an iterator to its position.
erase(k):	Remove the element with key k.
erase(p):	Remove the element referenced by iterator p.
begin():	Return an iterator to the beginning of the map.
end():	Return an iterator just past the end of the map.

```cpp
#include <iostream>
#include <map>
#include <string>

using namespace std;

int main(){

        map<string,int> m;

        m.insert(make_pair("a", 1));
        m.insert(make_pair("b", 2));
        m.insert(make_pair("c", 3));
        m.insert(make_pair("d", 4));
        m.insert(make_pair("e", 5));
        m["f"] = 6;


        m.erase("d");
        m.erase("e");
        m.erase(m.find("f"));

        if(!m.empty())
          cout << "m size : " << m.size() << '\n';

        cout << "a : " << m.find("a")->second << '\n';
        cout << "b : " << m.find("b")->second << '\n';

        cout << "a count : " << m.count("a") << '\n';
        cout << "b count : " << m.count("b") << '\n';

        for(std::map<string,int>::iterator it = m.begin(); it != m.end(); it++){
                cout << "key : " << it->first << " " << "value : " << it->second << '\n';
        }

        return 0;

}
```

```
m size : 3
a : 1
b : 2
a count : 1
b count : 1
key : a value : 1
key : b value : 2
key : c value : 3
```

پیاده‌سازی Map

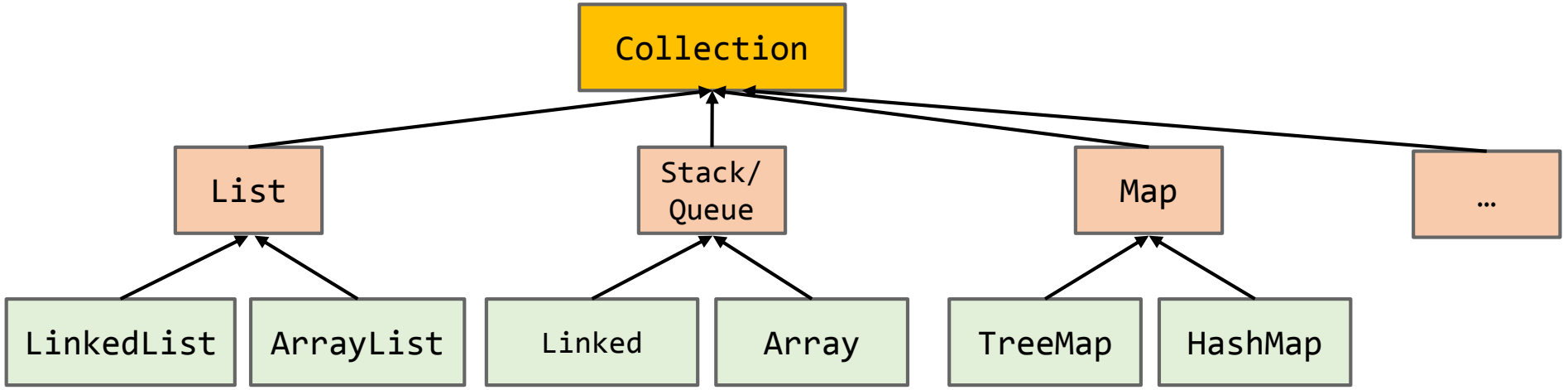
- برای یک map با n جفت (key, value)

	`insert`	`find`	`delete`
Unsorted linked-list	*O(1)*	*O(n)*	*O(n)*
Unsorted array	*O(1)*	*O(n)*	*O(n)*
Sorted linked list	*O(n)*	*O(n)*	*O(n)*
Sorted array	*O(n)*	*O(logn)*	*O(n)*
AVL/RB tree	*O(logn)*	*O(logn)*	*O(logn)*

پیاده‌سازی Map

- آیا برای زمانی که map به صورت unordered باشد پیاده‌سازی کارآمد داریم؟

ساختارهای داده



STL

- unordered_map

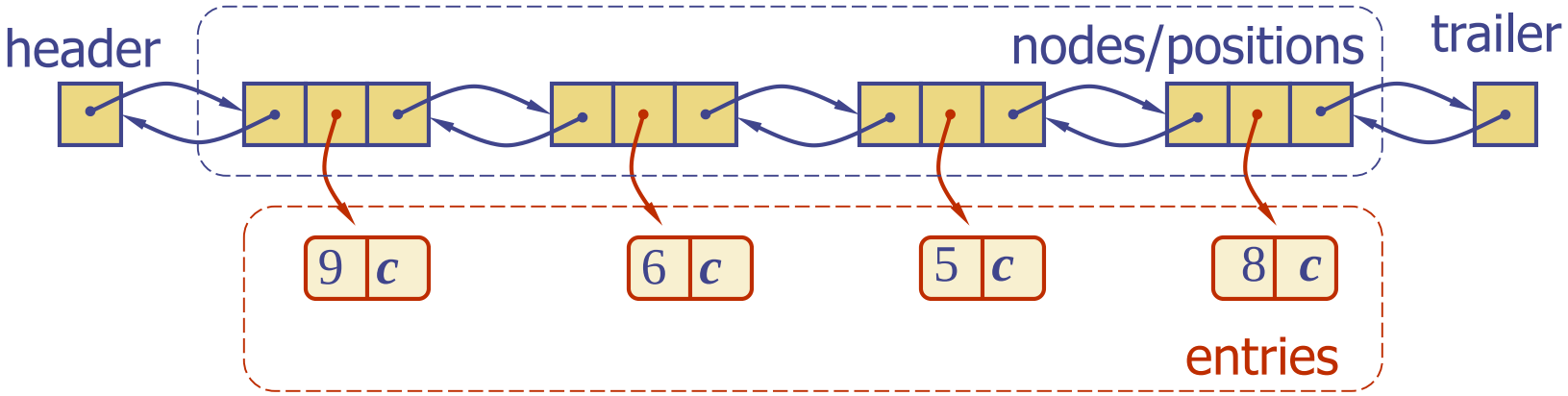
Recall the Map ADT

- find(k): if the map M has an entry with key k, return its associated value; else, return null
- put(k, v): insert entry (k, v) into the map M; if key k is not already in M, then return null; else, return old value associated with k
- erase(k): if the map M has an entry with key k, remove it from M and return its associated value; else, return null
- size(), empty()
- entrySet(): return a list of the entries in M
- keySet(): return a list of the keys in M
- values(): return a list of the values in M

- کدام ها در ordered map بودند و اینجا نیستند؟

A Simple List-Based Map

- An easiest way of implementing Map
- We can efficiently implement a map using an unsorted list
  - We store the items of the map in a list S (based on a doubly-linked list), in arbitrary order



The find Algorithm


```
Algorithm find(k):
    for each p in [S.begin(), S.end()) do
        if p→key() = k then
            return p
    return S.end() {there is no entry with key equal to k}
```

We use p→key() as a shortcut for (*p).key()

The put Algorithm


```
Algorithm put(k,v):
    for each p in [S.begin(), S.end()) do
        if p→key() = k then
            p→setValue(v)
            return p
    p = S.insertBack((k,v)) {there is no entry with key k}
    n = n + 1      {increment number of entries}
    return p
```

The erase Algorithm


```
Algorithm erase(k):
    for each p in [S.begin(), S.end()) do
        if p.key() = k  then
            S.erase(p)
            n = n – 1          {decrement number of entries}
```

Performance of a List-Based Map

- Performance:
  - put takes $O(n)$ time since we need to determine whether it is already in the sequence
  - find and erase take $O(n)$ time since in the worst case (the item is not found) we traverse the entire sequence to look for an item with the given key

- The unsorted list implementation is effective only for maps of small size or for maps in which puts are the most common operations, while searches and removals are rarely performed (e.g., historical record of logins to a workstation)

- Can we improve?

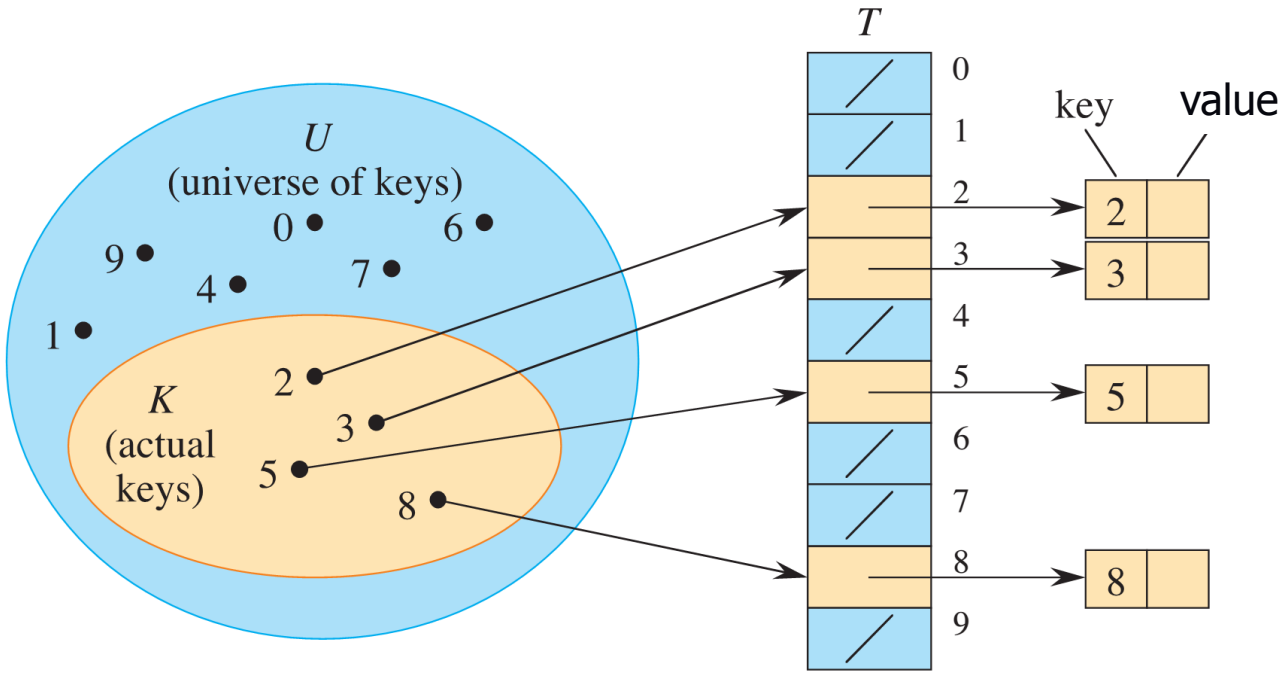
Hash Tables

What about this idea?

- example

Direct-address tables

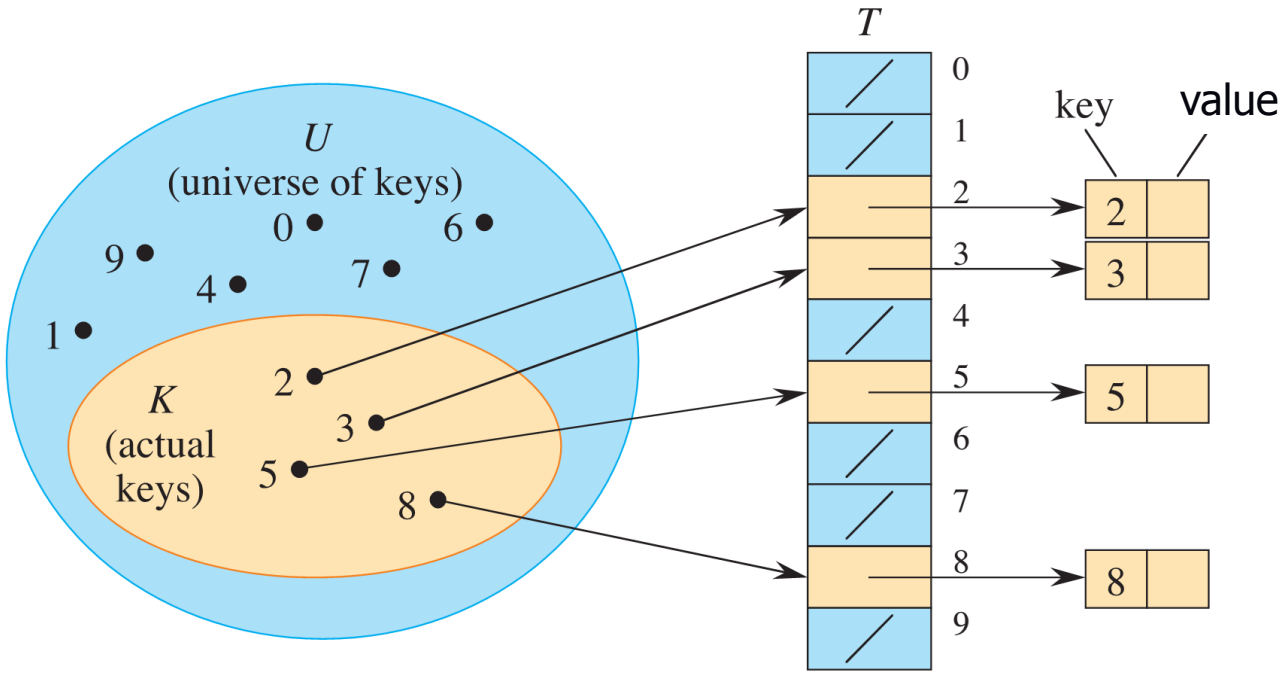
- you can use an array, or direct-address table, denoted by T, in which each position, or slot, corresponds to a key in the universe U .



Direct-address tables

- you can use an array, or direct-address table, denoted by T, in which each position, or slot, corresponds to a key in the universe U .

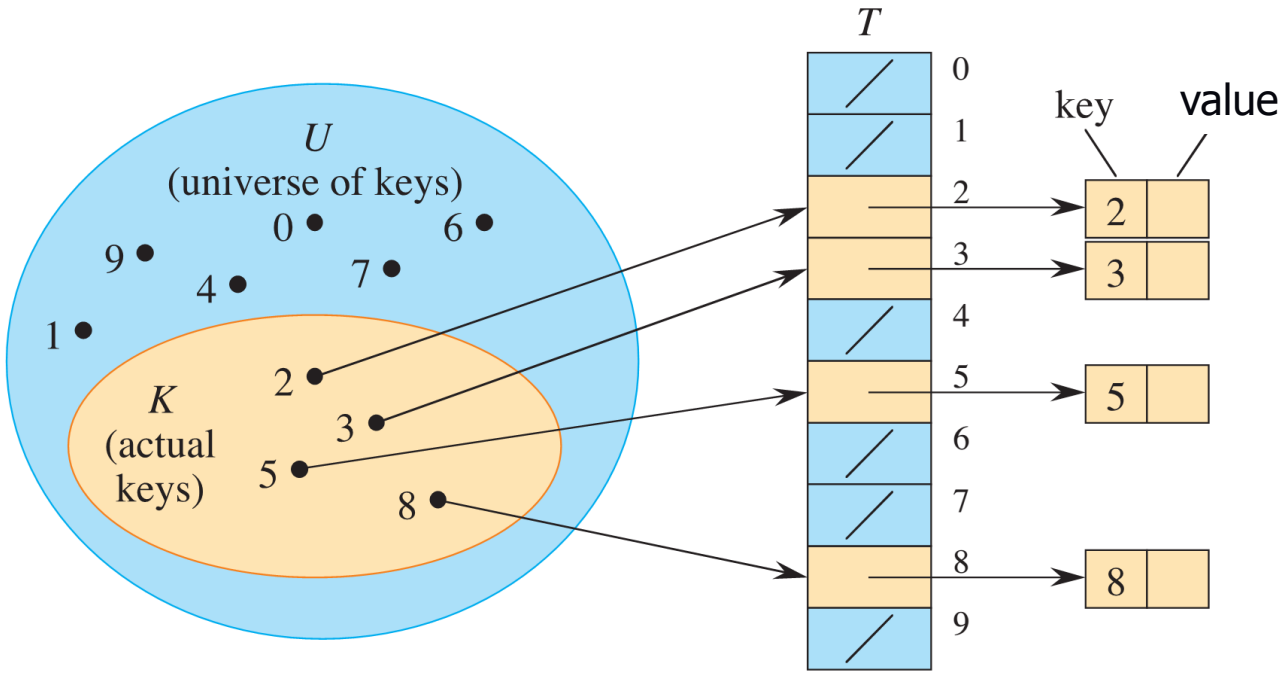
Search, Insert, Delete?



Direct-address tables

- you can use an array, or direct-address table, denoted by T, in which each position, or slot, corresponds to a key in the universe U .

Problem?

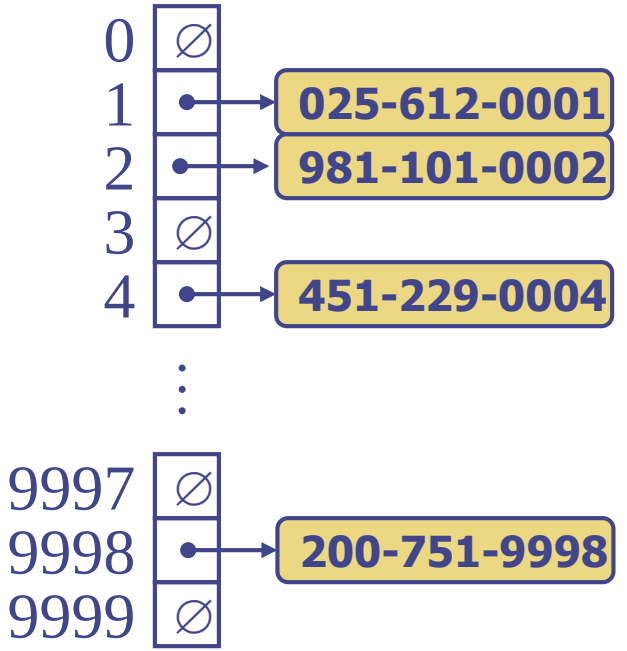


What about this idea?

- We design a table for a map storing entries as (SSN, Name), where SSN (social security number) is a nine-digit positive integer

- Our table uses an array of size $N = 10{,}000$ and classify each person based on the last four digits of his/her SSN

- ***Do you think that we can speed up searching using this method?***



Hash Table

- A hash table generalizes the simpler notion of an ordinary array.

Hash Table: Overview

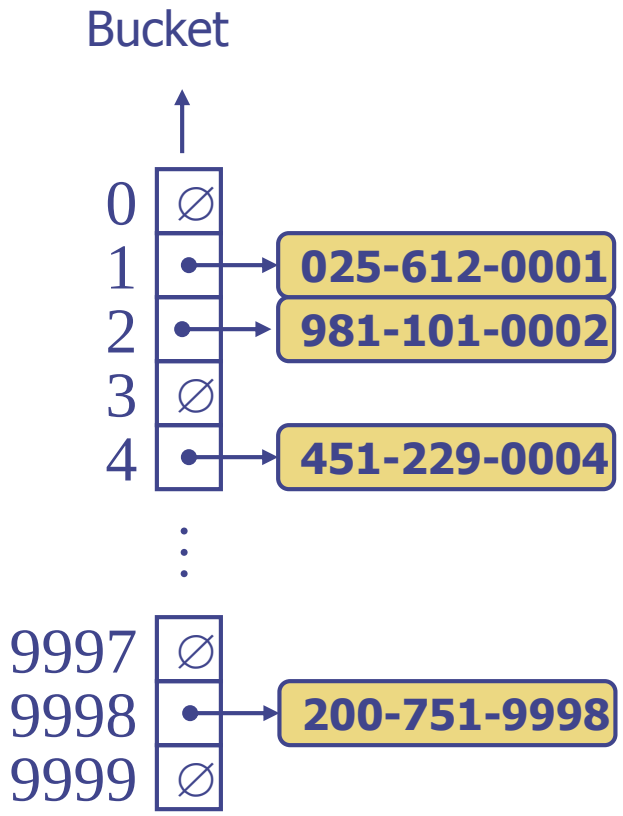
- Use key as an “address” for a value

- Worst-case performance: still O(n)
- But, usually expected performance: O(1)
  - Practically very fast

- Consists of two major components
  - 1. Bucket array
  - 2. Hash function

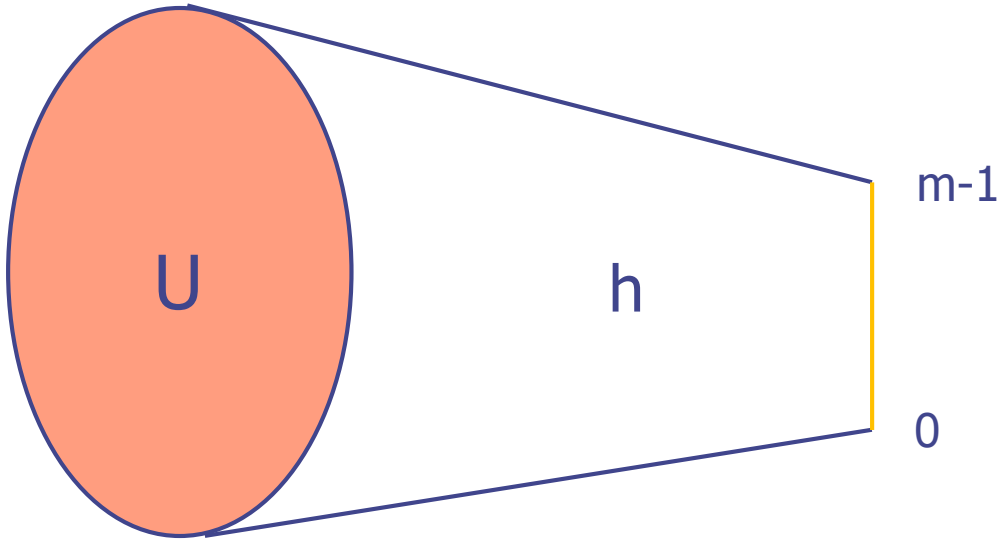
Bucket Array

- An array of size N, where each cell of A is “bucket”
- Two Issues
  - How to choose the bucket size N?
    - Large N?
    - Small N?
  - Keys should be integers, but in practice, not always.
    - Key: string “goodrich”



Hash Functions and Hash Tables

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



i=h(k)

Hash Functions and Hash Tables

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