



# IIT Madras

ONLINE DEGREE

**Programing: Data Structures and Algorithms using Python**  
**Professor Madhavan Mukund**  
**Implementation of Dictionaries in Python**

So, having seen lists and arrays in Python, let us look at another very popular data structure that we use quite often, which is built into Python, namely a dictionary. So, how are dictionaries actually implemented in Python?

(Refer Slide Time: 00:23)

Dictionary

- An array/list allows access through positional indices
- A dictionary allows access through arbitrary keys
  - A collection of key-value pairs
  - Random access — access time is the same for all keys
- How is a dictionary implemented?

Madhavan Mukund Implementing dictionaries

So, recall, that an array or a list allows us to access an element through its position. So, we can talk about  $l[i]$ , and  $i$  will range between 0 and  $n - 1$  where  $l$  is a list or an array of size  $n$ . So, the way we access an element or we name an element in a list or an array is through its position. If it is two-dimension, we will say row  $i$  column  $j$ .


In a dictionary, on the other hand, what we want is to name or access an element through some abstract value. For instance, we might want to attach, for instance, to name, a string, we might want to attach a value, for example, marks. We could have a dictionary, which has marks where the key is the name of the student or the roll number of the student, and the value that we store against that name, or roll number will be the marks of that student.

So, we do not want to go through this process of first creating a mapping from all students to  $0, 1, 2$  to  $n - 1$  and then using an array. So, we would directly like to use this key to value mapping, so we want what is called a key value store. But we want it to behave still like an array, in the sense that, if we pass a key to this dictionary we would like it to return the value in equal time for any key.

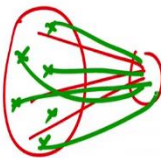
So, this is a principle of random access. It should not depend on what the key is, it should take roughly the same time to give me the value regardless of what key I provide. So, the question we want to know is, how is such a dictionary implemented.

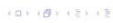
(Refer Slide Time: 01:48)

Implementing a dictionary




- The underlying storage is an array
  - Given an offset  $i$ , find  $A[i]$  in constant time
- Keys have to be mapped to  $\{0, 1, \dots, n-1\}$ 
  - Given an key  $k$ , convert it to an offset  $i$
- Hash function
  - $h: S \rightarrow X$  maps a set of values  $S$  to a small range of integers  $X = \{0, 1, \dots, n-1\}$
  - Typically  $|X| \ll |S|$ , so there will be collisions,  $h(s) = h(s'), s \neq s'$
  - A good hash function will minimize collisions





Madhavan Mukund

Implementing dictionaries



So, we know that, an array is the way to get random access memory. In an array, we have a contiguous block, and we can get the offset of position  $i$  by looking at the first block and multiplying it by the number of  $i$  times the value of the size of the data entry, and getting the  $i$ th block in constant time.

So, the underlying principle of any random access structure has to be this idea that you have a contiguous block and you can navigate an offset within the block. But our problem is that we want to not use the 0 to  $n$  minus 1 as our entries, but we want to use these keys, so we need a way to convert these keys into 0 to  $n$  minus 1.

So, we need a way to go from a key to a position in this array. So, we need a function, which can map keys to 0 to  $n$  minus 1. So, we want to give an a key  $k$  converted to an offset  $I$ , and this kind of a function is called a hash function. So, it takes a set of values. So, the set of values could be roll numbers, names, any set, and maps these values to a fixed range, typically zero to  $n$  minus 1.

So, it takes an arbitrary set and maps it to a small range of values. And usually, the range of values that you map it to, is going to be much smaller than the range of possibilities. So, you can imagine that if you think of all the possible names that you could have for students in a

class or all the possible roll numbers that you could create using the sequences of letters and numbers that you use for roll numbers that is a very large set.

But you will use a class will have only 50 people, 100 people, 200 people, whatever, some small number of actual people. So, you don't want to have an enormous possible storage for all possible names and all possible roll numbers, so you would actually like to map this large set of values to a small set.

But, of course, now the problem is, therefore, that if I look at all the possible names that people could come with or all the possible roll numbers I could create, in general, it will be that two of these keys will map to the same value, so this is what is called a collision. So, this hash function  $h$  maps  $s$  and  $s$  prime to the same value, even though it is not equal to  $s$  prime.

Now, this cannot be avoided, because you are taking a large set and you are pushing everything from here into a small set. So, it is not possible for everything in the large set to map to different things in the small set, because you just do not have that many elements to map to. But what you would like to do is, to somehow distribute this so that it is random.

So, it should not be that, if I map a name, and if I map the next name or if I map a roll number, and I map the next roll number the next roll number will go and collide as it is called, so these are called collisions. So, the next number will collide. I would like to distribute, so that if I give you a, so the ideal thing is that if I give you a random set of elements from here, which are kind of spaced out, hopefully they will all go to different things, this is what you would like.

You cannot guarantee it, but you would like this to happen. So, if I give you a random collection of inputs from the last set, then with reasonable chance they should not collide. So, this is what a good hash function will do.

(Refer Slide Time: 04:58)

The slide is titled "Implementing a dictionary" and features a list of bullet points. Handwritten green notes are present: "256", "25", "256 bits", and "1000". A video inset shows a man in a blue shirt speaking. The slide footer includes the name "Madhavan Mukund" and the title "Implementing dictionaries".

- The underlying storage is an array
  - Given an offset  $i$ , find  $A[i]$  in constant time
- Keys have to be mapped to  $\{0, 1, \dots, n-1\}$ 
  - Given an key  $k$ , convert it to an offset  $i$
- Hash function
  - $h: S \rightarrow X$  maps a set of values  $S$  to a small range of integers  $X = \{0, 1, \dots, n-1\}$
  - Typically  $|X| \ll |S|$ , so there will be collisions,  $h(s) = h(s')$ ,  $s \neq s'$
  - A good hash function will minimize collisions
  - SHA-256 is an industry standard hashing function whose range is 256 bits
    - Use to hash large files — avoid uploading duplicates to cloud storage

Just to illustrate, how important hash functions are. So, there is this now, current industry standard called SHA-256. So, this is used for cryptography and other things. So, 256 refers to the fact that the output of this thing is to 256 bits. 256 bits is a large number it is 2 to the power 256. So, remember that 2 to the power 10 is 1,000. So, this is 1,000 to the power 25, roughly something like that.

So, this is like 1 with 75 zeros after it. So, that is the space in which you are mapping. But it is not such a large space, if you think about it, because what you can actually do is you can take, for instance an entire file. Now, a file could be a very large. So, you can think of a file as a large number, a file is a sequence of zeros and ones, so you can think of it as a large number or some value from some large set, and you can now map it to this 256 bit.

So, for instance, it could be a movie or it could be a document. So, it could be a very large file much larger than this 1 with 75 zeros in terms of possibilities. So, each individual document will not be so much, but there are so many different documents. So, one application of SHA-256, which might be surprising to you, is that, if you do if you use a cloud storage system, supposing something like Dropbox, for instance.

Sometimes, when you upload a very large file, it will actually upload very fast. And the reason is that actually, this system computes this SHA-256 hash. And if it detects that this

hash is already present in the storage, it does not actually upload, it just kind of makes a pointer saying one more copy of this file has been uploaded, it does not bother to upload it.

So, in principle, if you had a hash, SHA-256, hash with two different files produce the same hash, you would not be able to upload the second file because it will say it is already there. Now, in practice, this does not happen, so it just shows that the SHA-256, has been carefully designed so that these collisions rarely happen.

(Refer Slide Time: 07:11)

Hash table

- An array  $A$  of size  $n$  combined with a hash function  $h$
- $h$  maps keys to  $\{0, 1, \dots, n-1\}$
- Ideally, when we create an entry for key  $k$ ,  $A[h(k)]$  will be unused

$d[k]=v$

$k \rightarrow h(k)$  no  $j$  in  $0-n$

Madhavan Mukund Implementing dictionaries

So, the way we use this hash function, as we said, is we want to take the key and map it to a position in an array, so this is called a hash table. So, a hash table is an array of size  $n$ , combined with a hash function, which maps keys to this range  $0$  to  $n$  minus  $1$ . So, the idea is that  $h$  will map the keys that you provide, whatever keys they are, so the keys are not restricted, only the range of the hash function is restricted, it depends on what the keys are.

So, the ideal situation is that if I give you a new key,  $k$ , and I want you to store something as for the dictionary,  $d$  of  $k$ , then you will map it to this  $h$  of  $k$  the position in the array and hopefully that places available to you. So, I want to say  $d$  of  $k$  is equal to  $V$ .

So, what will happen now, is that, I will go from  $k$  to  $h$  of  $k$ , this will give me a number. This will be some number in  $0$  to  $n$  minus  $1$ . Then I will go into my array, so this will be some number,  $j$ . So, then I will go to position  $j$ , and I will try to put this value  $V$  there. So, this is how dictionary should work. But what will happen is that sometimes because of collisions, that place is occupied.



(Refer Slide Time: 08:24)

Hash table

- An array  $A$  of size  $n$  combined with a hash function  $h$
- $h$  maps keys to  $\{0, 1, \dots, n-1\}$
- Ideally, when we create an entry for key  $k$ ,  $A[h(k)]$  will be unused
  - What if there is already a value at that location?
- Dealing with collisions
  - Open addressing (closed hashing)
    - Probe a sequence of alternate slots in the same array

Madhavan Mukund Implementing dictionaries

So, if there is already a value at that location then this hash table has to deal with this collision. It cannot just replace because then their old value will be lost. It cannot put it in some random place, because if it puts it in a random place, you will never find it. So, you need to have a systematic way of putting it somewhere where you will find it if you look at it later. So, there are two ways to deal with this, which have some more confusing names.

So, the first one is called open addressing, but it is called closed hashing. And here, the idea is, you could not go outside the array. So, what you have is that you came here. So, this is the place that you were looking to fill, but then unfortunately for you there is already something here. So now what you do is, you skip, right, so you kind of have a kind of round robin. So, you do not necessarily go to the next position.

But you kind of say, okay, if this is not there then let me go plus 10 and see if there is a place there. If it is not there, then let me go plus 10. So, I just keep cycling around, going forward by probing a sequence of alternative slots until I find a free slot. And in that free slot, now I must of course have one extra piece of work, I must write the key also because I need to know that this piece belongs to that key. But essentially, this is the way you work, so you create an array which is large enough, so there will be some extra slots.

So, even after these collisions, you will still find some extra space. So, if you anticipate that you will need say, 1,000 entries in your dictionary, then you might put say an array of size

5,000. So, you have a lot of free space to play with for this kind of collision. hashing, hashing and avoiding collisions.

(Refer Slide Time: 10:04)

Hash table

- An array  $A$  of size  $n$  combined with a hash function  $h$
- $h$  maps keys to  $\{0, 1, \dots, n-1\}$
- Ideally, when we create an entry for key  $k$ ,  $A[h(k)]$  will be unused
  - What if there is already a value at that location?
- Dealing with collisions
  - Open addressing (closed hashing)
    - Probe a sequence of alternate slots in the same array
  - Open hashing
    - Each slot in the array points to a list of values
    - Insert into the list for the given slot

Madhavan Mukund Implementing dictionaries

The other option is to do what is called open hashing. So, this is now the confusion. Open hashing and open addressing are not the same. So, open addressing means you are working within the same array. You are moving around to a new position and hoping it is free. And so it is like you get into, say, supposing you get into a parking lot, and you find that there is no space for you to park in that particular row.

Then you go out and you circle around and you enter another row, and you try to look for a space, then you go out circle. Eventually, hopefully, there is a space, you will find it. So, that is more like closed hashing. You just stay within that parking lot. You are looking for a free slot, so the free slot, this first slot you try to find was booked, so you go around looking for the next free slot by randomly wandering around, not quite randomly.

But you move around in some fixed sequence so that you can retrace that sequence later on when somebody comes looking for this value again. Now in open hashing, on the other hand, remember that you get into this, and this is my array. So, in close hashing, I actually store the value in the array. Now in open hashing, what I do is I do not store the value in the array instead, for each position in the array, I maintain a list.

So, the actual storage happens somewhere outside this hash table. So, the hash table is only a collection of what you might call pointers. So, if I hash key to a position in the array, the



array tells me, go and look there, that is the list of all values, which had the same hash value as the key you had, go and find it. So, I will maintain this elsewhere.

And then now because that list is flexible, you can grow and shrink, I do not have any constraint about it. So, this allows me to keep coalitions under control in that sense, because I can just append to that list. So, these are these two standard ways. And, of course, you can use some heuristics to do these better or worse, but this is the standard ways of dealing with collisions in a hash table.

(Refer Slide Time: 11:52)

Hash table

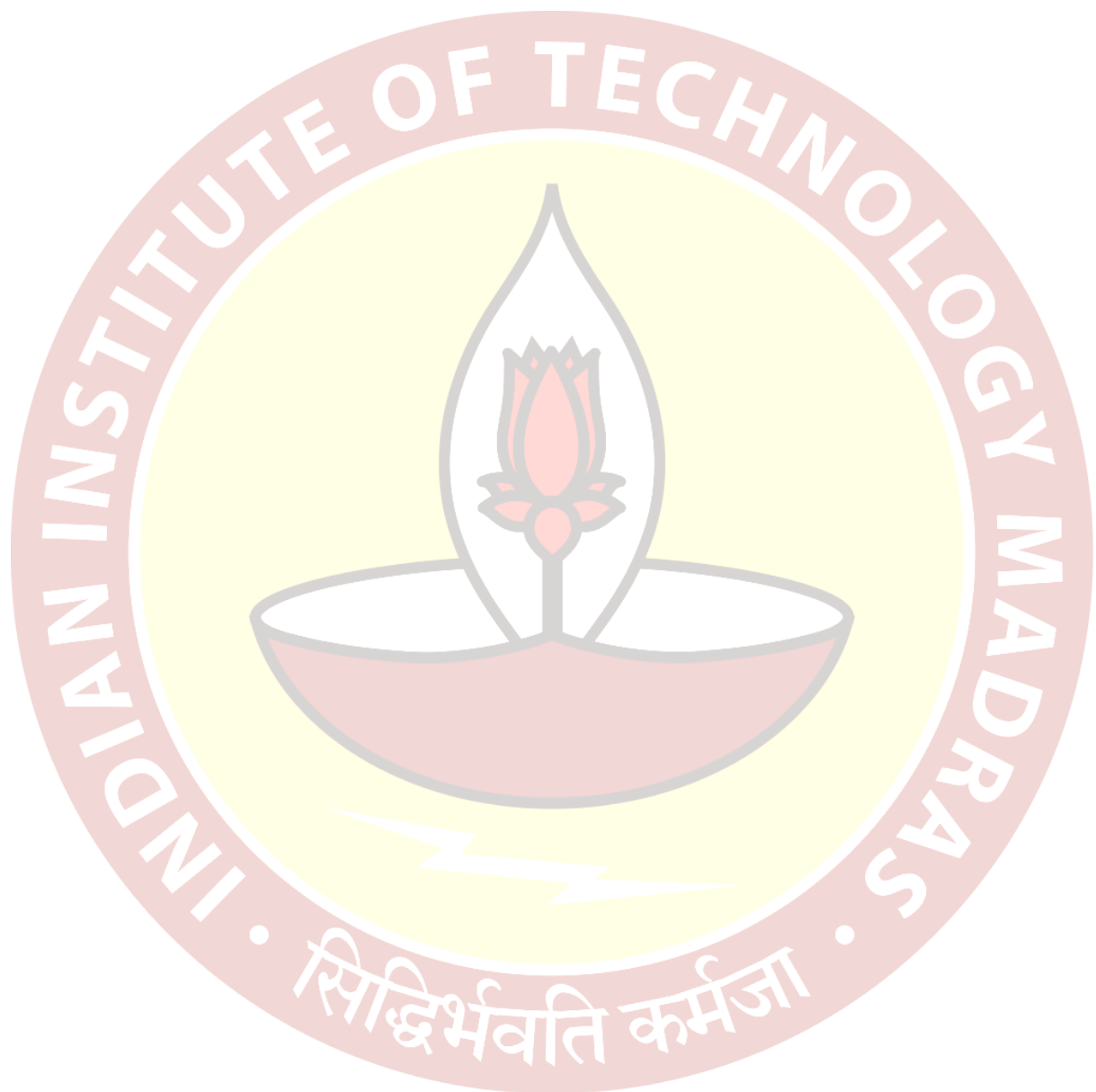
- An array  $A$  of size  $n$  combined with a hash function  $h$
- $h$  maps keys to  $\{0, 1, \dots, n-1\}$
- Ideally, when we create an entry for key  $k$ ,  $A[h(k)]$  will be unused
  - What if there is already a value at that location?
- Dealing with collisions
  - Open addressing (closed hashing)
    - Probe a sequence of alternate slots in the same array
  - Open hashing
    - Each slot in the array points to a list of values
    - Insert into the list for the given slot
- Dictionary keys in Python must be immutable
  - If value changes, hash also changes!

Madhavan Mukund Implementing dictionaries

So, you might have wondered. When you are taught to use dictionaries in Python, one of the things you are told is the key to a dictionary must be immutable. You cannot use for example, a list as a key for a dictionary. Why is that the case? Well, now you know the answer. Because, in order to find the place where the value is stored, the hash function has to be applied. So, the hash function is applied, when it is stored, at that point, the key has a value.

Now, if it is a mutable data structure, the value can change. So, when I come back later on, and I try to hash that value, it will be a new value, and therefore I will not be able to find that position in the dictionary anymore. So, I must guarantee that the value that I used inside the key remains fixed, because otherwise the hash function cannot return the same value in a guaranteed way each time.

So, this is one reason why Python insists that dictionary keys must be immutable, so that the hash value remains fixed across multiple visits to that key when you enter that same dictionary.



(Refer Slide Time: 12:58)

Summary

- A dictionary is implemented as a hash table
  - An array plus a hash function
- Creating a good hash function is important (and hard!)
- Need a strategy to deal with collisions
  - Open addressing/closed hashing — probe for free space in the array
  - Open hashing — each slot in the hash table points to a list of key-value pairs
  - Many heuristics/optimizations possible for dea

$h: k \rightarrow i \in [0, n-1]$

Madhavan Mukund Implementing dictionaries

So, to summarize, dictionaries are extremely powerful data structures, which allow you to store values against keys. And the way you implement a dictionary is by actually using this hash function. So, underlying it, there is an array of some sort, which has a random access storage from zero to  $n$  minus 1, but you want to take a key and you want to put it at position  $i$ .

So, you want to take key and you want to map it to  $i$  belonging to 0 to  $n$  minus 1. And this is that function,  $h$ , which is a hashing function. So, you want to design a good hash function, which randomizes and does not collide too often. So, creating a good hash function is a lot of work. So, the SHA-256, which I mentioned, was a major effort.

It was a major effort to come up with this function, which would give you an output or 256 bits, and which had all the desirable properties. So, to come up with a good hash function requires a lot of mathematical reasoning about the function, and we are not going to get into that, so we assume that we have a hash function.

And in fact, in Python, there are hash functions. There are hash functions for different immutable types, like strings and numbers and all that. So, you can actually create your own hash table if you really want by using the Python hashing function. But when we use a hash table, one of the things that we need to do is to deal with collisions.

Because it is eminently possible that at different point in time, two keys will give you the same location inside that array that you are using. So, either you can use this probing

technique, which is to just move around the array looking for a free space, which is called open addressing, or closed hashing depending on what you want to call it.

So, think of it as closed hashing means your world is closed, you cannot move outside this area. So, you are restricted to being inside this array. Whereas open hashing says actually, the place that you are going to go to is somewhere outside. So, each of the slots in the array actually points to a list of values, which shared the same position.

So, if you have a collision, you just move and append to that list, so each position  $i$  now maps to a collection of values. So, you first come here and then go there, and it still does not take much time. Because you assume, again, that you have a good hash, so these collisions are rare. So, those lists will be ideally small. You do not want everything to be collecting in one list, which would happen if all your elements collide.

So, if you have a reasonable hash function, it will distribute the values across these lists, so searching those lists will not take much extra time. So, there are many heuristics and optimizations which are available for dealing with collisions which we will not get into.

