hi guys welcome back in this video we're building something very interesting we're building our own cache using golang so you must have heard about redis and hazelcast and the way they uh you know save time and save operations by not letting your back end call your database again and again and caching and storing that information for small periods of time so that's what we're building so we'll try to build um a full-on cache but uh we uh won't be building the operations a lot of operations are required to you know maintain a cache and a lot of we'll have to get into a lot of algorithms but we'll be using just data structures we'll just build the cache uh and we'll build some operations and adding and deleting from the cache but we won't build the advanced operations that are required the algorithms are required in the cache this is an intermediate level video it's not an advanced video it's not a beginner level video at all so uh if you're very comfortable with golang very comfortable with pointers and structs and all those different concepts this is the right video for you so you'll be able to get to build your own lru least recently used cache using uh linked lists uh which is the data structure in golang all right so uh if you can see my screen well so i'll take you through the entire project as in what we're about to do but let me show you what it looks like so if i run so i'll say go and main.go what happens here is it it's taking some values like cat blue dog tree and dragon it's creating this uh cube out of it q and the q is the cache in our case and um let's let's not uh you know uh get bogged down with all the all this that you see on the screen let me first explain to you what's going to happen so we'll have a cache i've demonstrated that using this kind of uh uh you know like a rectangle so it maintains a specific length length of items so this length of items we can always keep changing based on our preferences but let's say it maintains a specific length of items and it's called the lru cache only the most recently used items would be kept in that gag in that cache so there's an lru cache and then there's a true lru cache if you read about it more on the internet that for a true lru cache which we're going to build uh there are these three conditions that need to be met the first is if an item is if an item already exists in that cache we need to remove it and add it again to the beginning so let's say the cache is there and already has a lot of values so uh again that value uh comes into the cache that that means you have to check and then you have to find that value in that cache you have to remove it and then add this same value in the beginning of the cache so that it stays least recently used right it stays recently used and you can access it so because all the old items are going to be removed anyways and the second um condition is an order of the items needs to be maintained you can't just store them uh what do you call it uh like a like a big data object without any order you have to have an order of these items so that's another criteria that needs to be met the third is the deletion happens at the tail mostly so we can delete uh items from the from between the cache also which i'll show you uh how to do that and but mostly the deletion happens only at the tail and the addition to the cache happens only at the head so there'll be head to this cache there'll be a tail to this cache like we'll have a queue and we'll have a head and a tail so and so on from this side uh the addition you can see the item goes in and but from this side the end the item goes out when the capacity is full full right the capacity obviously depends on the specific length of the items that we specify for that cache anyhow now the cache is made up of a queue and the queue is made up of multiple nodes this is very similar actually is the same concept as a linked list so in a linked list what happens is you have a node and then you have a left and a right and the left and also this node at the in the middle now the left stores the address to another node right which will be which will be basically on the left so whatever node will be on the left right of this list to create the queue of this node to

create the queue the the left variable will store the address and the right variable will show the address of the right uh the left and right nodes respectively all right now it's quite possible that you still aren't understanding much it's not a problem when we'll start coding everything will start making a lot more sense to you and then we have a concept called as the queue now queue has two three things one is the head one is the tail and the other is the length length is the integer of the number of items in the queue and head is basically an empty node uh so this node is going to be extract right it is going to have three things left node and right and then node itself is going to be obstruct and then the queue is going to be a struct in our project which will have a head which will be a node and a tail which would be a node and these will be empty and these will just contain the head will always contain or always point to the first element in the queue and the tail will always point to the last element in the queue so the tail uh since it's a node as you know whenever you look at tail it's a node you already know that it has left node and right these three things that means the left of the tail will always point to the last element in the queue and the right of the head right of the head will always point to the first element in the queue all right so that's that forms up our queue now we uh in our cache along with our queue we also have one more thing called the hash so hash is a map basically and it has a text text will be the values that we'll send it's all these values that you'll send like blue cat all of these things these will be uh the text or the strings and then um they will have a node associated to it so we're just maintaining all of that in a hash basically in the cache so uh if you want to add something to your queue right so like i said you know it will only go from uh it will only be added here at the beginning of the queue which is the head of the queue to add something what you have to do is you have to point the right of this particular node the new node that's coming in to the first element of the existing queue and the left of this node will point to the head of the queue and then automatically this new element will be added to the queue all right again i'm saying that if it's not making much sense if it's all looking very confusing when we'll code uh we'll go through these diagrams again again and everything will start making a lot more sense to you now to delete uh an item from the queue so we have to take into account two different scenarios one is that the element is deleted from the last uh last place which is the tail or the element is deleted somewhere in between so uh like i said you know i said that the deletion happens at the tail and addition happens in the beginning so in the beginning the addition happens and when we'll exceed the limit of the cache the last element of the cache will be removed right as as the node so when it's removed from the as a node it will be removed at the tail at the end but then sometimes what also happen what's happening here is that uh like i said the the criteria for the least recently used cache is that if an item exists we need to remove it and add it to the beginning so the item could exist somewhere in the between in between the queue and we have to remove it and add it to the beginning of the queue so uh so we'll have to have a scenario that also helps us to uh delete items from the middle of the queue as well so what we'll have to do is we'll have to basically come up with a very common kind of uh formula or a way to basically remove elements from the cache so what we're going to do basically is uh whenever we have to remove some node from the cache we'll point our right uh sorry uh yeah so we'll take that node like this node has to be removed all right so this uh the the node to the left of this node that has to be removed the right of this node will point to uh what this node node's write is pointing to which is basically this node right node three so i'll say one two and three to make it more clear so the nodes um so the first node's right needs to be pointing at what the second node's right is

pointing to currently [Music] and the third node's left needs to be pointing to what the second node's left is pointing to currently so that means if this points to this and this points to this then automatically second gets deleted because nobody has address of second anymore so automatically it's deleted from memory so that's going to be a deletion now again like i said you know it might not make much sense to you maybe you're very new to data structures you have no idea what link lists are not a problem we'll uh go through them one by one again again so what i want to do now is i want to uh create a project all right so i'll say i'll come back to my workspace and i'll create a new project i'll say mkdir and i'll say cache project all right so we'll see it into it and uh we'll also say go mod init github.com slash akil slash cash project go mod in it if you are new to golang uh by the way if you need to go long and this is probably not the right video for you but anyhow if you're new to golang and uh go if you're seeing this video for the first time then go mod in it basically is like if you're coming from a javascript background sorry if you're from a javascript background then go modern it is like uh npm in it you know it creates a package.json kind of a file for you so it created a go mod file which is like package.json file basically and what we'll do is so this file the go mod file creates uh you know holds a list of all your dependencies of your project in our project i don't think we'll have any external dependencies so let's go ahead and open this project let me bring it onto the screen so that you can see it and here what we'll have to do is we'll have to create our main.co file where at the beginning we'll say package main and then we'll import a couple of things for now let's just import fmt and then let's create our funk main so in our funk main all we're doing is we're just writing fmt dot println start cache and we're creating a variable called cache and we'll have a function called new cache we'll just initiate the new cache for us and then what we're going to do is uh we're going to say [Music] we'll use a for loop and we'll take a slice of strings and here we can enter our input so all the element that we want to add to our cache all right and then there will be so all these all these elements will be here inside this slice of strings and then we'll go over them one by one and then all of those values can be accessed using uh this value called word and then we want to have a function called check and i'll explain to you why we need this function and then we have another function called display all right so these two functions are what you need to keep in mind and here you'll just enter the list of all the things that you want to enter into the cache so for example we want to say uh parrot and avocado i'll just put some random values all right but avocado and we will say dragon fruit and let's say um i'm not uh tree [Music] and potato tomato [Music] we'll say tree again [Music] and dog any random stuff anyways so those are the things that you want your cache to process and store all right now let's create try and create like an outline of the project that based on the diagram that i showed you there needs to be an outline of the entire project so that means that we'll have a struct called node okay so let me show it on the diagram also so the struct call node is this it has something always left and right and it has a node all right so that will be our struct called node and then you'll have a start called q then you'll have [Music] a hash map which will be string and it will have a node associated to it so it's capital n because that's what we've written here so q and hash map right so as you can see we have a queue which will have a head tail and uh integer length and we have a hash map which basically has text which you just saw string and then uh it'll have a address of the node so that's what we've done till now okay and then we have our cache so like i told you the cache [Music] has two things it is basically q of type q and it has a hash of type hash quite straightforward and then we're calling this function called new

cache so we'll have to have that function called new cache function new cache and it will return a cache and then we'll have a function so inside this function right we're just going to return a cache which will basically be a queue where we'll have a function called new queue to create the queue for us and we'll have a hash an empty hash right so hash can obviously um call this struct is type of destruct right hash that's an empty struct of type hash that's what hash is and q is going to be created in a function called neoq so now let's create uh let's work on that function so we'll say funk new q [Music] all right and we'll have something here we'll have the queue here so now let's go one by one and start creating all of our structs so the node struct has a value of type string which basically is the value here [Music] so instead of node you have to write value so it'll have a value here of type string he'll have a left which will reference a node it will have a right which will reference a node okay so on the left we'll have referencing a node points to node and the right points to a node to the right all right so that's what we're storing in one particular node that's what you do where we're doing so our node is complete now our head the rq has a head which is of type node and has a tail of type node and it has a length of type end and where did we see that before we this is the queue it has a head of type node so it has an empty node that points to the first element of the cube which we'll get that to that later on it has a tail which is an empty node and then it has high length all right so far so good [Music] now i think we should work on our new queue function so a new queue function takes a variable called head [Music] and takes an empty struct of type node it has a tail again the address of an empty node okay and then we'll take head so i'll write something and then i'll explain to you what's happening here it's very very simple actually [Music] okay so let me draw it for you actually [Music] let me draw the diagram for you so that you'll understand more visually basically so if you have no elements here [Music] and only the head and tail are present so we'll say head and [Music] we'll say tail right so in the queue we have these two empty things head and tail now the right of this head if it points towards the tail and the and the left of the tail points towards the right that means that the queue is empty because there are no values in between them because the head and tail will always be empty and head and tail we're just maintaining to uh basically uh you know always the head will point to the first element of the queue and the tail will point to the last element of the queue that's why we're maintaining these head and tail in a queue so if they point to each other then that means that q is empty right and that's how you initialize a cube so that's what's happening here so the right of the head is pointing to the tail and the tails left is pointing to the head very simple so all we have to do is return q here the head of which is the head and the tail of which is the tail [Music] all right now if you notice something in a queue the head has a capital h and a capital t which basically you're returning and uh but it's equal to the small uh h head and the small etched tail small detail right so which is basically what we're defining here so that means we have met the criteria of this being the queue and we have created a proper queue and that's what we're determining from the new queue function which goes to the new cache function and you know it takes the queue and the hash at the same time and that's what's happening and now comes the most important part where we have to start working on our uh check and display functions now display function is quite simple so we can do that at the end and i think we should work on the check function all right so let's build the check function so we'll say funk [Music] cash all right because it's a stuck method for cash if you don't know much about stuck methods then i suggest you check them out they're an important concept in golang so here we'll take a variable called node and we'll you know make it empty start of type node

and what you're doing here is uh let me how to explain to you it's actually quite simple so this check uh function right what it does is it just basically takes all these values one by one because we're running a for loop and these values one by one will go to this function called check now what we want to do in the check function is we want to see if that value already exists in that queue if it exists in that queue we want to remove that value and add it at the beginning if it doesn't exist we just want to add that value to the queue that's all this check function is doing all right uh and why do we want to do why do we want to delete a value if it already exists and add it again it's because this is the least recently used cache so the values if if that values was used uh much before we want it to be since it's being used right now so we wanted to come to the beginning because only the recently used values need to stay in the cache and the older ones will get deleted okay so that's what we're doing the check function it's very very straightforward so we'll say since we're maintaining a hash in the cache right we'll say [Music] if that value is present in the hash map so the value is basically the string the string that you'll pass from this for loop you'll pass the strings one by one to the check method check method okay which is of type struct cache struct method and in this uh method you'll get the string and the string you're checking in your hash in your cache is hash you're checking if the string exists if there's a node associated with the string and if everything is okay then what you do is you say node c dot remove so like i said you know if it's already there in that cache you remove that value and if it's not there you just take a variable called node and you make the value of that node equal to the string so as you remember every node has something called as a value so that value will become now equal to the string if it wasn't there in the uh what you call it in the hash if it was there in the hash you deleted it if it was not there in the hash you just created a node where the value is equal to the string okay and all you have to do is c dot add node you just add the node to it and then you also add that value to the hash so we'll say string equal to node okay so here uh by mistake i've used the walrus operator you just have to use a normal director regular is equal to so it's giving you these squiggly lines because remove function is not there and the add function is not there but we've already seen using this diagram how addition and deletion needs to take place right so let me explain to you again uh the node that needs to be added to this queue the right of that node will start pointing the first element that was existing in this queue and the left of this node will point to the uh the head of the queue and while deletion is the same right whatever it's not the same actually because we want to be able to delete from the from the you know from between the values as well so uh the left of uh one will so sorry the right of one will point to the right of two and the right theft of three will point to the left off too right so that's what uh we will do to delete a value so it's going to be pretty simple [Music] all right remove and add functions are going to be pretty simple as well so let's go ahead and create our remove function first so we'll say funk c cache and remove n and it it basically turns a node after removing it so once you remove it it basically returns your node and you're going to add that node once it's removed right so that's why we're taking this value into node and if it wasn't there in the hash then it's just going to uh you know create a node with that value which string the string that you had passed and at the end the basic thing that's going to happen is going to add that node to the cache [Music] so let's write our remove function so we'll say fmt dot printf remove percentage s new line so print f where there's an f like sprint f or printf it helps you also to format your string so this percentage s will be uh replaced by n dot value n is the node that we are replacing sorry are removing so the value of that will be printed here all right [Music] and

[Music] here we'll also have left is equal to n dot left we'll take two uh variables left and right and we'll point them to where this node that has to be removed to the left and right of that particular node so we'll say n dot right [Music] okay [Music] and [Music] we'll say right dot left is equal to left [Music] sorry this is a capital r right dot left is left and left dot right is equal to right [Music] so what's happening here is pretty straight forward like i've already explained that the node that has to be removed the left of that you take into a variable called left and the right of that you take into a variable called right and the left and the right of that left variable left points to the right and the right points to the left so the [Music] so f2 is the uh node to be deleted then one's right is going to point to two's right where two's right was pointing and third three is left well point to where the two's left was pointing which is one right so one and three will start pointing to each other and two will disappear that's happening basically here okay straight forward and we'll reduce [Music] the length [Music] by one of the queue [Music] and we'll use a delete function which is a neater function with poland and we'll say c dot hash and dot value basically you remove that value from the hash and you return n [Music] okay [Music] now we have two errors one is for the add function now that is for the display function so let's build our add and display function as well so we'll say funk c star cache and okay so we're passing a node to the add function and we'll say fmt dot print f [Music] and dot value [Music] we'll take a temporary variable [Music] and we'll point it to uh the current queues uh where the head of the current q's right is pointing so which will be basically the first element so that we have taken into temp all right so we'll say c dot q dot add [Music] dot right so the new um place where the head of the queue will point right now will be the new node that we have passed so what's happening is that we pass a node to be added to the add function and then this uh this node now basically uh is what the head of the queue is going to point towards all right and then the current first element which is this where the heads right was pointing to this is the current first element before we had added this node will be taken in this variable called temp okay so we'll say n dot left is equal to c dot q dot head and n dot write [Music] is equal to temp variable and temp dot left is equal to n okay so what's happened here is [Music] the queues uh till here we're understood right so let me show it to you again so with addition what happens is that the right of this node needs to point to the first element that was already existing in the queue so this element we took into this variable called temp all right so we took this variable this value the first node and the queue into a value called temp then we took the current head of the queue and the right of that we pointed to this first element that will be now the first element right this is a node that we have said that will be added to this queue so the right at the head of this uh queue will now point to this instead of this right and so this element into this node it's going to point to this node all right and the nodes left will point to the head and the nodes right will point to temp so what's happening here is the nodes left will point to the head element of the queue which is the first or the head element of the queue and the nodes right this new node's right will point to the first l element which was previously the first element right so this will not truly become the first element of the queue okay so straightforward if you uh you know if you've not understood this and if you're finding this confusing you can always put that in the comments below and i'll try to explain to you what i'm trying to do is i'm trying to explain every single line by going uh and showing the diagram for every single line and uh i'm not sure but i'm still not sure if i'm able to uh you know explain it to people who don't have a background with data structures because resources are quite important [Music] i hope i'm able to explain it but if i'm if i'm doing a bad job then let me know

and i'll try to explain to you on a personal call if you want and what we'll do is c dot q dot length so if you add a add an element to the cache right you will have to add or you'll have to increase the length and if you would we were deleting something from the cache we were reducing the length so here you're increasing the length and we're doing one more thing so while adding something to the cache we're also checking that if the the you know the size of the cache has gone beyond what we had defined so what we'll do is we'll take a take a variable called size so after our import we'll say constant size is equal to five so this means that this is the size of our cache the size of our cache needs to be five and here what we are saying is in our uh new add add new function we are saying that if this the length of the queue q dot length is greater than size so we have extended we've gone beyond the size that was given to us then we have to remove something from the end so we'll say c dot remove remove the last element which is c dot q dot tail dot left [Music] so the tail of the queue uh the left of the tier of the queue is pointing to the last element as you know as you recall so that last element of the queue needs to be removed if we have already exceeded the size of the queue after this addition function has happened now all we are left to do is create this display function so this display function basically takes a cache right so we'll say funk dash display [Music] sorry this display function is of type cache it's a cache method so we'll say c dot q dot display so to display the queue now qr display doesn't uh exist right so we'll have to create it so we'll have to create a display method for the struct uh which is uh q so let's say q star q [Music] display so this display method is for is of type cache struct and this is for q struct right so we have two different structs for cache and queue so basically our cache display method calls our queued stream so that's what this is doing and so we'll take a node it's a q dot head dot right [Music] so this means the the the right with the where the head of your queue the right of that is pointing that is obviously the first element in your queue right so you take that in node and then you say fmp dot printf percentage d [Music] and this is where we'll start and comma q dot length so we'll print out the length out there the length of the queue is prevented first and then we'll say for i not equal to zero i less than q dot length i plus plus [Music] so we'll run the for loop we'll say fmt dot printf percentage s [Music] so this basically helps us to print the node nodes value between those uh curly braces as you'd seen in the beginning of the video if i less than q dot length minus 1 [Music] then we'll say fmt dot printf [Music] [Music] so node digital node.right helps you to point to the next element right so one by one by one you can keep pointing the next element so you since you're in a for loop right so node is when node becomes no dot right that means right and then right and then right again again you can keep going to the next element very easily [Music] and here you'll say fmt dot print ln so these brackets this bracket and this bracket you'll understand once i once we run this code it's just for formatting and these curly braces are also for formatting and then this percentage s basically gets replaced by node.value the value gets printed here one by one and uh that's that's about it so nothing else is happening here and let's try running this code and there could be errors which i'm pretty sure there will be errors so let's say go run dot co [Music] and wow there are no errors and everything works perfectly and uh what's happening here is that first you added padded so the length is printed out here the length of the node is printed out here [Music] this one the packet is printed and then the length of the model is printed so let's see the bracket is printed and the lower length of the node is printed and then inside that bracket we have our parrot the first element which is no dot value right so one by one by one it'll keep going through all the uh you know uh till the whole complete length of the queue and print

it out and here the length of the queue becomes two and then it prints out these two values length of the queue becomes three but it brings up these three values and what's happening here it's adding tomato and removing parrot [Music] because uh when you add potato right you reach uh the length which is five which is the maximum length allowed to this cache but when you try to add tomato it will have to remove the last value from this cache right from this queue so we'll remove parrot from there so now you have these values now as you can see that uh tree is already a value that exists so when you try to send tree here again so it'll remove tree and then it'll uh again add tree to it so then the latest elements are maintained basically then you add dog and remove avocado avocados the last element you remove it and then you add dog to it and that's when you get this basically and that's how our queue is working let me take you through the entire code again so what's happening in our funk main is now we start our cache and then we have a list of these values that we want to put into our queue and one by one by one uh since we have a for loop running uh one by one all these values uh will be iterated through and then we'll call this function called check and then we'll call this function called display and both of these functions are of type these are methods for the cache struct so for the check function what happens is you go to check function in the check function you create a variable called node which is an empty struct of type node and you check if that value that string that has come into this function already exists in our hash for the cache uh now the cache basically create consists of a queue and a hash basically right so in the hash we're checking if that value exists if it exists already then you remove it if it doesn't exist then you create that try and create that value but uh even after you've removed that value you want to add that value right so if it didn't exist then you also you'll add it or if it existed then you remove it and then add it in the beginning so the add function all it does is it adds uh any value to the beginning of the queue and the remove function basically removes a value from the queue and display basically displays it uh very simply right so every single time you're adding something to the queue or deleting something to the queue you're printing out or displaying the entire queue again and again and that's what's happening here right so every single time the display function will be called now this uh uh node node has a value a left and right which you've understood so every single node will have a left which will point to the left another node the right will point to another node and the queue will have a head and tail which are empty nodes and the head basically does the the work of pointing to the first value of the queue which will be a node and the tail does the value of the work of pointing to the last value of the queue which will be you know again a node right so that's what basically our entire project is now it's quite uh if if you were already familiar with uh linked lists then everything would be much simpler to you but in case you don't have an idea about what linked lists are and how data structures work then obviously all of this will be very difficult for you and if you didn't understand anything in this project then do let me know and i'll try and explain uh it to you uh if you have any issues in the comments and do let me know so thanks a lot for watching and and on this channel we'll have a lot more videos coming up with kolank so and i'll see you in the next episode and do stay subscribed thank you hi guys welcome back in this video we're building something very interesting we're building our own cache using golang so you must have heard about redis and hazelcast and the way they uh you know save time and save operations by not letting your back end call your database again and again and caching and storing that information for small periods of time so that's what we're building so we'll try to build um a full-on cache but uh we uh won't be building the

operations a lot of operations are required to you know maintain a cache and a lot of we'll have to get into a lot of algorithms but we'll be using just data structures we'll just build the cache uh and we'll build some operations and adding and deleting from the cache but we won't build the advanced operations that are required the algorithms are required in the cache this is an intermediate level video it's not an advanced video it's not a beginner level video at all so uh if you're very comfortable with golang very comfortable with pointers and structs and all those different concepts this is the right video for you so you'll be able to get to build your own lru least recently used cache using uh linked lists uh which is the data structure in golang all right so uh if you can see my screen well so i'll take you through the entire project as in what we're about to do but let me show you what it looks like so if i run so i'll say go and main.go what happens here is it it's taking some values like cat blue dog tree and dragon it's creating this uh cube out of it q and the q is the cache in our case and um let's let's not uh you know uh get bogged down with all the all this that you see on the screen let me first explain to you what's going to happen so we'll have a cache i've demonstrated that using this kind of uh uh you know like a rectangle so it maintains a specific length length of items so this length of items we can always keep changing based on our preferences but let's say it maintains a specific length of items and it's called the lru cache only the most recently used items would be kept in that gag in that cache so there's an lru cache and then there's a true lru cache if you read about it more on the internet that for a true lru cache which we're going to build uh there are these three conditions that need to be met the first is if an item is if an item already exists in that cache we need to remove it and add it again to the beginning so let's say the cache is there and already has a lot of values so uh again that value uh comes into the cache that that means you have to check and then you have to find that value in that cache you have to remove it and then add this same value in the beginning of the cache so that it stays least recently used right it stays recently used and you can access it so because all the old items are going to be removed anyways and the second um condition is an order of the items needs to be maintained you can't just store them uh what do you call it uh like a like a big data object without any order you have to have an order of these items so that's another criteria that needs to be met the third is the deletion happens at the tail mostly so we can delete uh items from the from between the cache also which i'll show you uh how to do that and but mostly the deletion happens only at the tail and the addition to the cache happens only at the head so there'll be head to this cache there'll be a tail to this cache like we'll have a queue and we'll have a head and a tail so and so on from this side uh the addition you can see the item goes in and but from this side the end the item goes out when the capacity is full full right the capacity obviously depends on the specific length of the items that we specify for that cache anyhow now the cache is made up of a queue and the queue is made up of multiple nodes this is very similar actually is the same concept as a linked list so in a linked list what happens is you have a node and then you have a left and a right and the left and also this node at the in the middle now the left stores the address to another node right which will be which will be basically on the left so whatever node will be on the left right of this list to create the queue of this node to create the queue the the left variable will store the address and the right variable will show the address of the right uh the left and right nodes respectively all right now it's quite possible that you still aren't understanding much it's not a problem when we'll start coding everything will start making a lot more sense to you and then we have a concept called as the queue now queue has two three things one is the head one is the tail and the other is the length length is the

integer of the number of items in the queue and head is basically an empty node uh so this node is going to be extract right it is going to have three things left node and right and then node itself is going to be obstruct and then the queue is going to be a struct in our project which will have a head which will be a node and a tail which would be a node and these will be empty and these will just contain the head will always contain or always point to the first element in the queue and the tail will always point to the last element in the queue so the tail uh since it's a node as you know whenever you look at tail it's a node you already know that it has left node and right these three things that means the left of the tail will always point to the last element in the queue and the right of the head right of the head will always point to the first element in the queue all right so that's that forms up our queue now we uh in our cache along with our queue we also have one more thing called the hash so hash is a map basically and it has a text text will be the values that we'll send it's all these values that you'll send like blue cat all of these things these will be uh the text or the strings and then um they will have a node associated to it so we're just maintaining all of that in a hash basically in the cache so uh if you want to add something to your queue right so like i said you know it will only go from uh it will only be added here at the beginning of the queue which is the head of the queue to add something what you have to do is you have to point the right of this particular node the new node that's coming in to the first element of the existing queue and the left of this node will point to the head of the queue and then automatically this new element will be added to the queue all right again i'm saying that if it's not making much sense if it's all looking very confusing when we'll code uh we'll go through these diagrams again again and everything will start making a lot more sense to you now to delete uh an item from the queue so we have to take into account two different scenarios one is that the element is deleted from the last uh last place which is the tail or the element is deleted somewhere in between so uh like i said you know i said that the deletion happens at the tail and addition happens in the beginning so in the beginning the addition happens and when we'll exceed the limit of the cache the last element of the cache will be removed right as as the node so when it's removed from the as a node it will be removed at the tail at the end but then sometimes what also happen what's happening here is that uh like i said the the criteria for the least recently used cache is that if an item exists we need to remove it and add it to the beginning so the item could exist somewhere in the between in between the queue and we have to remove it and add it to the beginning of the queue so uh so we'll have to have a scenario that also helps us to uh delete items from the middle of the queue as well so what we'll have to do is we'll have to basically come up with a very common kind of uh formula or a way to basically remove elements from the cache so what we're going to do basically is uh whenever we have to remove some node from the cache we'll point our right uh sorry uh yeah so we'll take that node like this node has to be removed all right so this uh the the node to the left of this node that has to be removed the right of this node will point to uh what this node node's write is pointing to which is basically this node right node three so i'll say one two and three to make it more clear so the nodes um so the first node's right needs to be pointing at what the second node's right is pointing to currently [Music] and the third node's left needs to be pointing to what the second node's left is pointing to currently so that means if this points to this and this points to this then automatically second gets deleted because nobody has address of second anymore so automatically it's deleted from memory so that's going to be a deletion now again like i said you know it might not make much sense to you maybe you're very new to data structures you have

no idea what link lists are not a problem we'll uh go through them one by one again again so what i want to do now is i want to uh create a project all right so i'll say i'll come back to my workspace and i'll create a new project i'll say mkdir and i'll say cache project all right so we'll see it into it and uh we'll also say go mod init github.com slash akil slash cash project go mod in it if you are new to golang uh by the way if you need to go long and this is probably not the right video for you but anyhow if you're new to golang and uh go if you're seeing this video for the first time then go mod in it basically is like if you're coming from a javascript background sorry if you're from a javascript background then go modern it is like uh npm in it you know it creates a package.json kind of a file for you so it created a go mod file which is like package.json file basically and what we'll do is so this file the go mod file creates uh you know holds a list of all your dependencies of your project in our project i don't think we'll have any external dependencies so let's go ahead and open this project let me bring it onto the screen so that you can see it and here what we'll have to do is we'll have to create our main.co file where at the beginning we'll say package main and then we'll import a couple of things for now let's just import fmt and then let's create our funk main so in our funk main all we're doing is we're just writing fmt dot println start cache and we're creating a variable called cache and we'll have a function called new cache we'll just initiate the new cache for us and then what we're going to do is uh we're going to say [Music] we'll use a for loop and we'll take a slice of strings and here we can enter our input so all the element that we want to add to our cache all right and then there will be so all these all these elements will be here inside this slice of strings and then we'll go over them one by one and then all of those values can be accessed using uh this value called word and then we want to have a function called check and i'll explain to you why we need this function and then we have another function called display all right so these two functions are what you need to keep in mind and here you'll just enter the list of all the things that you want to enter into the cache so for example we want to say uh parrot and avocado i'll just put some random values all right but avocado and we will say dragon fruit and let's say um i'm not uh tree [Music] and potato tomato [Music] we'll say tree again [Music] and dog any random stuff anyways so those are the things that you want your cache to process and store all right now let's create try and create like an outline of the project that based on the diagram that i showed you there needs to be an outline of the entire project so that means that we'll have a struct called node okay so let me show it on the diagram also so the struct call node is this it has something always left and right and it has a node all right so that will be our struct called node and then you'll have a start called q then you'll have [Music] a hash map which will be string and it will have a node associated to it so it's capital n because that's what we've written here so q and hash map right so as you can see we have a queue which will have a head tail and uh integer length and we have a hash map which basically has text which you just saw string and then uh it'll have a address of the node so that's what we've done till now okay and then we have our cache so like i told you the cache [Music] has two things it is basically q of type q and it has a hash of type hash quite straightforward and then we're calling this function called new cache so we'll have to have that function called new cache function new cache and it will return a cache and then we'll have a function so inside this function right we're just going to return a cache which will basically be a queue where we'll have a function called new queue to create the queue for us and we'll have a hash an empty hash right so hash can obviously um call this struct is type of destruct right hash that's an empty struct of type hash that's what hash is and q

is going to be created in a function called neoq so now let's create uh let's work on that function so we'll say funk new q [Music] all right and we'll have something here we'll have the queue here so now let's go one by one and start creating all of our structs so the node struct has a value of type string which basically is the value here [Music] so instead of node you have to write value so it'll have a value here of type string he'll have a left which will reference a node it will have a right which will reference a node okay so on the left we'll have referencing a node points to node and the right points to a node to the right all right so that's what we're storing in one particular node that's what you do where we're doing so our node is complete now our head the rq has a head which is of type node and has a tail of type node and it has a length of type end and where did we see that before we this is the queue it has a head of type node so it has an empty node that points to the first element of the cube which we'll get that to that later on it has a tail which is an empty node and then it has high length all right so far so good [Music] now i think we should work on our new queue function so a new queue function takes a variable called head [Music] and takes an empty struct of type node it has a tail again the address of an empty node okay and then we'll take head so i'll write something and then i'll explain to you what's happening here it's very very simple actually [Music] okay so let me draw it for you actually [Music] let me draw the diagram for you so that you'll understand more visually basically so if you have no elements here [Music] and only the head and tail are present so we'll say head and [Music] we'll say tail right so in the queue we have these two empty things head and tail now the right of this head if it points towards the tail and the and the left of the tail points towards the right that means that the queue is empty because there are no values in between them because the head and tail will always be empty and head and tail we're just maintaining to uh basically uh you know always the head will point to the first element of the queue and the tail will point to the last element of the queue that's why we're maintaining these head and tail in a queue so if they point to each other then that means that q is empty right and that's how you initialize a cube so that's what's happening here so the right of the head is pointing to the tail and the tails left is pointing to the head very simple so all we have to do is return q here the head of which is the head and the tail of which is the tail [Music] all right now if you notice something in a queue the head has a capital h and a capital t which basically you're returning and uh but it's equal to the small uh h head and the small etched tail small detail right so which is basically what we're defining here so that means we have met the criteria of this being the queue and we have created a proper queue and that's what we're determining from the new queue function which goes to the new cache function and you know it takes the queue and the hash at the same time and that's what's happening and now comes the most important part where we have to start working on our uh check and display functions now display function is quite simple so we can do that at the end and i think we should work on the check function all right so let's build the check function so we'll say funk [Music] cash all right because it's a stuck method for cash if you don't know much about stuck methods then i suggest you check them out they're an important concept in golang so here we'll take a variable called node and we'll you know make it empty start of type node and what you're doing here is uh let me how to explain to you it's actually quite simple so this check uh function right what it does is it just basically takes all these values one by one because we're running a for loop and these values one by one will go to this function called check now what we want to do in the check function is we want to see if that value already exists in that queue if it exists in that queue we want to remove that value and add it at the beginning if it

doesn't exist we just want to add that value to the queue that's all this check function is doing all right uh and why do we want to do why do we want to delete a value if it already exists and add it again it's because this is the least recently used cache so the values if if that values was used uh much before we want it to be since it's being used right now so we wanted to come to the beginning because only the recently used values need to stay in the cache and the older ones will get deleted okay so that's what we're doing the check function it's very very straightforward so we'll say since we're maintaining a hash in the cache right we'll say [Music] if that value is present in the hash map so the value is basically the string the string that you'll pass from this for loop you'll pass the strings one by one to the check method check method okay which is of type struct cache struct method and in this uh method you'll get the string and the string you're checking in your hash in your cache is hash you're checking if the string exists if there's a node associated with the string and if everything is okay then what you do is you say node c dot remove so like i said you know if it's already there in that cache you remove that value and if it's not there you just take a variable called node and you make the value of that node equal to the string so as you remember every node has something called as a value so that value will become now equal to the string if it wasn't there in the uh what you call it in the hash if it was there in the hash you deleted it if it was not there in the hash you just created a node where the value is equal to the string okay and all you have to do is c dot add node you just add the node to it and then you also add that value to the hash so we'll say string equal to node okay so here uh by mistake i've used the walrus operator you just have to use a normal director regular is equal to so it's giving you these squiggly lines because remove function is not there and the add function is not there but we've already seen using this diagram how addition and deletion needs to take place right so let me explain to you again uh the node that needs to be added to this queue the right of that node will start pointing the first element that was existing in this queue and the left of this node will point to the uh the head of the queue and while deletion is the same right whatever it's not the same actually because we want to be able to delete from the from the you know from between the values as well so uh the left of uh one will so sorry the right of one will point to the right of two and the right theft of three will point to the left off too right so that's what uh we will do to delete a value so it's going to be pretty simple [Music] all right remove and add functions are going to be pretty simple as well so let's go ahead and create our remove function first so we'll say funk c cache and remove n and it it basically turns a node after removing it so once you remove it it basically returns your node and you're going to add that node once it's removed right so that's why we're taking this value into node and if it wasn't there in the hash then it's just going to uh you know create a node with that value which string the string that you had passed and at the end the basic thing that's going to happen is going to add that node to the cache [Music] so let's write our remove function so we'll say fmt dot printf remove percentage s new line so print f where there's an f like sprint f or printf it helps you also to format your string so this percentage s will be uh replaced by n dot value n is the node that we are replacing sorry are removing so the value of that will be printed here all right [Music] and [Music] here we'll also have left is equal to n dot left we'll take two uh variables left and right and we'll point them to where this node that has to be removed to the left and right of that particular node so we'll say n dot right [Music] okay [Music] and [Music] we'll say right dot left is equal to left [Music] sorry this is a capital r right dot left is left and left dot right is equal to right [Music] so what's happening here is pretty straight forward like i've already explained that the node that has

to be removed the left of that you take into a variable called left and the right of that you take into a variable called right and the left and the right of that left variable left points to the right and the right points to the left so the [Music] so f2 is the uh node to be deleted then one's right is going to point to two's right where two's right was pointing and third three is left well point to where the two's left was pointing which is one right so one and three will start pointing to each other and two will disappear that's happening basically here okay straight forward and we'll reduce [Music] the length [Music] by one of the queue [Music] and we'll use a delete function which is a neater function with poland and we'll say c dot hash and dot value basically you remove that value from the hash and you return n [Music] okay [Music] now we have two errors one is for the add function now that is for the display function so let's build our add and display function as well so we'll say funk c star cache and okay so we're passing a node to the add function and we'll say fmt dot print f [Music] and dot value [Music] we'll take a temporary variable [Music] and we'll point it to uh the current queues uh where the head of the current q's right is pointing so which will be basically the first element so that we have taken into temp all right so we'll say c dot q dot add [Music] dot right so the new um place where the head of the queue will point right now will be the new node that we have passed so what's happening is that we pass a node to be added to the add function and then this uh this node now basically uh is what the head of the queue is going to point towards all right and then the current first element which is this where the heads right was pointing to this is the current first element before we had added this node will be taken in this variable called temp okay so we'll say n dot left is equal to c dot q dot head and n dot write [Music] is equal to temp variable and temp dot left is equal to n okay so what's happened here is [Music] the queues uh till here we're understood right so let me show it to you again so with addition what happens is that the right of this node needs to point to the first element that was already existing in the queue so this element we took into this variable called temp all right so we took this variable this value the first node and the queue into a value called temp then we took the current head of the queue and the right of that we pointed to this first element that will be now the first element right this is a node that we have said that will be added to this queue so the right at the head of this uh queue will now point to this instead of this right and so this element into this node it's going to point to this node all right and the nodes left will point to the head and the nodes right will point to temp so what's happening here is the nodes left will point to the head element of the queue which is the first or the head element of the queue and the nodes right this new node's right will point to the first l element which was previously the first element right so this will not truly become the first element of the queue okay so straightforward if you uh you know if you've not understood this and if you're finding this confusing you can always put that in the comments below and i'll try to explain to you what i'm trying to do is i'm trying to explain every single line by going uh and showing the diagram for every single line and uh i'm not sure but i'm still not sure if i'm able to uh you know explain it to people who don't have a background with data structures because resources are quite important [Music] i hope i'm able to explain it but if i'm if i'm doing a bad job then let me know and i'll try to explain to you on a personal call if you want and what we'll do is c dot q dot length so if you add a add an element to the cache right you will have to add or you'll have to increase the length and if you would we were deleting something from the cache we were reducing the length so here you're increasing the length and we're doing one more thing so while adding something to the cache we're also checking that if the the you know the size of the cache has

gone beyond what we had defined so what we'll do is we'll take a take a variable called size so after our import we'll say constant size is equal to five so this means that this is the size of our cache the size of our cache needs to be five and here what we are saying is in our uh new add add new function we are saying that if this the length of the queue q dot length is greater than size so we have extended we've gone beyond the size that was given to us then we have to remove something from the end so we'll say c dot remove remove the last element which is c dot q dot tail dot left [Music] so the tail of the queue uh the left of the tier of the queue is pointing to the last element as you know as you recall so that last element of the queue needs to be removed if we have already exceeded the size of the queue after this addition function has happened now all we are left to do is create this display function so this display function basically takes a cache right so we'll say funk dash display [Music] sorry this display function is of type cache it's a cache method so we'll say c dot q dot display so to display the queue now qr display doesn't uh exist right so we'll have to create it so we'll have to create a display method for the struct uh which is uh q so let's say q star q [Music] display so this display method is for is of type cache struct and this is for q struct right so we have two different structs for cache and queue so basically our cache display method calls our queued stream so that's what this is doing and so we'll take a node it's a q dot head dot right [Music] so this means the the the right with the where the head of your queue the right of that is pointing that is obviously the first element in your queue right so you take that in node and then you say fmp dot printf percentage d [Music] and this is where we'll start and comma q dot length so we'll print out the length out there the length of the queue is prevented first and then we'll say for i not equal to zero i less than q dot length i plus plus [Music] so we'll run the for loop we'll say fmt dot printf percentage s [Music] so this basically helps us to print the node nodes value between those uh curly braces as you'd seen in the beginning of the video if i less than q dot length minus 1 [Music] then we'll say fmt dot printf [Music] [Music] so node digital node.right helps you to point to the next element right so one by one by one you can keep pointing the next element so you since you're in a for loop right so node is when node becomes no dot right that means right and then right and then right again again you can keep going to the next element very easily [Music] and here you'll say fmt dot print ln so these brackets this bracket and this bracket you'll understand once i once we run this code it's just for formatting and these curly braces are also for formatting and then this percentage s basically gets replaced by node.value the value gets printed here one by one and uh that's that's about it so nothing else is happening here and let's try running this code and there could be errors which i'm pretty sure there will be errors so let's say go run dot co [Music] and wow there are no errors and everything works perfectly and uh what's happening here is that first you added padded so the length is printed out here the length of the node is printed out here [Music] this one the packet is printed and then the length of the model is printed so let's see the bracket is printed and the lower length of the node is printed and then inside that bracket we have our parrot the first element which is no dot value right so one by one by one it'll keep going through all the uh you know uh till the whole complete length of the queue and print it out and here the length of the queue becomes two and then it prints out these two values length of the queue becomes three but it brings up these three values and what's happening here it's adding tomato and removing parrot [Music] because uh when you add potato right you reach uh the length which is five which is the maximum length allowed to this cache but when you try to add tomato it will have to remove the last value from this cache right from this queue

so we'll remove parrot from there so now you have these values now as you can see that uh tree is already a value that exists so when you try to send tree here again so it'll remove tree and then it'll uh again add tree to it so then the latest elements are maintained basically then you add dog and remove avocado avocados the last element you remove it and then you add dog to it and that's when you get this basically and that's how our queue is working let me take you through the entire code again so what's happening in our funk main is now we start our cache and then we have a list of these values that we want to put into our queue and one by one by one uh since we have a for loop running uh one by one all these values uh will be iterated through and then we'll call this function called check and then we'll call this function called display and both of these functions are of type these are methods for the cache struct so for the check function what happens is you go to check function in the check function you create a variable called node which is an empty struct of type node and you check if that value that string that has come into this function already exists in our hash for the cache uh now the cache basically create consists of a queue and a hash basically right so in the hash we're checking if that value exists if it exists already then you remove it if it doesn't exist then you create that try and create that value but uh even after you've removed that value you want to add that value right so if it didn't exist then you also you'll add it or if it existed then you remove it and then add it in the beginning so the add function all it does is it adds uh any value to the beginning of the queue and the remove function basically removes a value from the queue and display basically displays it uh very simply right so every single time you're adding something to the queue or deleting something to the queue you're printing out or displaying the entire queue again and again and that's what's happening here right so every single time the display function will be called now this uh uh node node has a value a left and right which you've understood so every single node will have a left which will point to the left another node the right will point to another node and the queue will have a head and tail which are empty nodes and the head basically does the the work of pointing to the first value of the queue which will be a node and the tail does the value of the work of pointing to the last value of the queue which will be you know again a node right so that's what basically our entire project is now it's quite uh if if you were already familiar with uh linked lists then everything would be much simpler to you but in case you don't have an idea about what linked lists are and how data structures work then obviously all of this will be very difficult for you and if you didn't understand anything in this project then do let me know and i'll try and explain uh it to you uh if you have any issues in the comments and do let me know so thanks a lot for watching and and on this channel we'll have a lot more videos coming up with kolank so and i'll see you in the next episode and do stay subscribed thank you