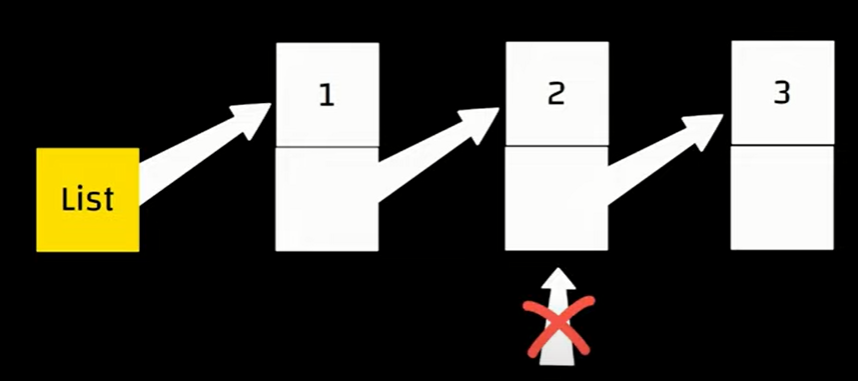
***Data Structure Tree***

*Humans always like the trees , their structure and composition , and it’s was the source of inspiration for many poets and scholars , and know you learn tree data structure , the wonderfull compostion that start from root then branches then leafs , and believe me tree is like data stucture tree, the neutral tree begin with root then the root grow branch out into branch and also branch grow and grow then the leafs start appearing , and also data structure tree start from root then root branch and each branch branch also .*

*As we know data structure is a way or it’s manner to save data in memory , or simply it’s manner to store data in memory in order of the abbility for arriving to it when we need , by example if you store* the data in memory by a sequential manner like array , you will get many features and speed access is the important feature . by example you want to get a specific element in array you have only to enter its index number , and this grant us a big feature which is binary search , but array has also many problem and among them the inability for resizing an array and for slove this problem we decide to use linked list , as we know linked list didn’t store the data in memory by a sequential manner like array but it group data with pointer for make us an element named node .

So the process is the first node point on the second node and vice versa … so linked list slove resize problem that we find it in array .

But on the other hand , another problem appear which is abbilty to access linked list element only from the begging point or the head .

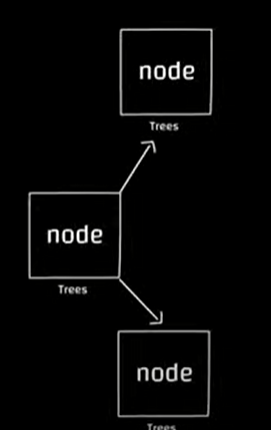
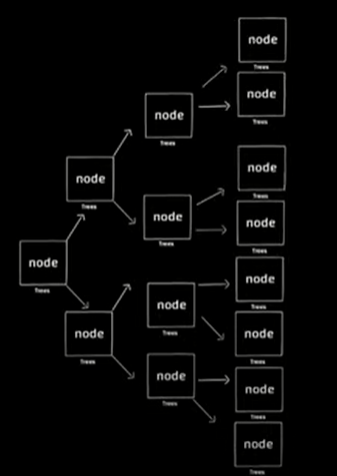
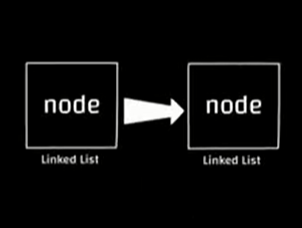
So if a want to access to the third element , so i have to enter by the starting point then cross all element by a linear manner , and this is the opposit of the array .

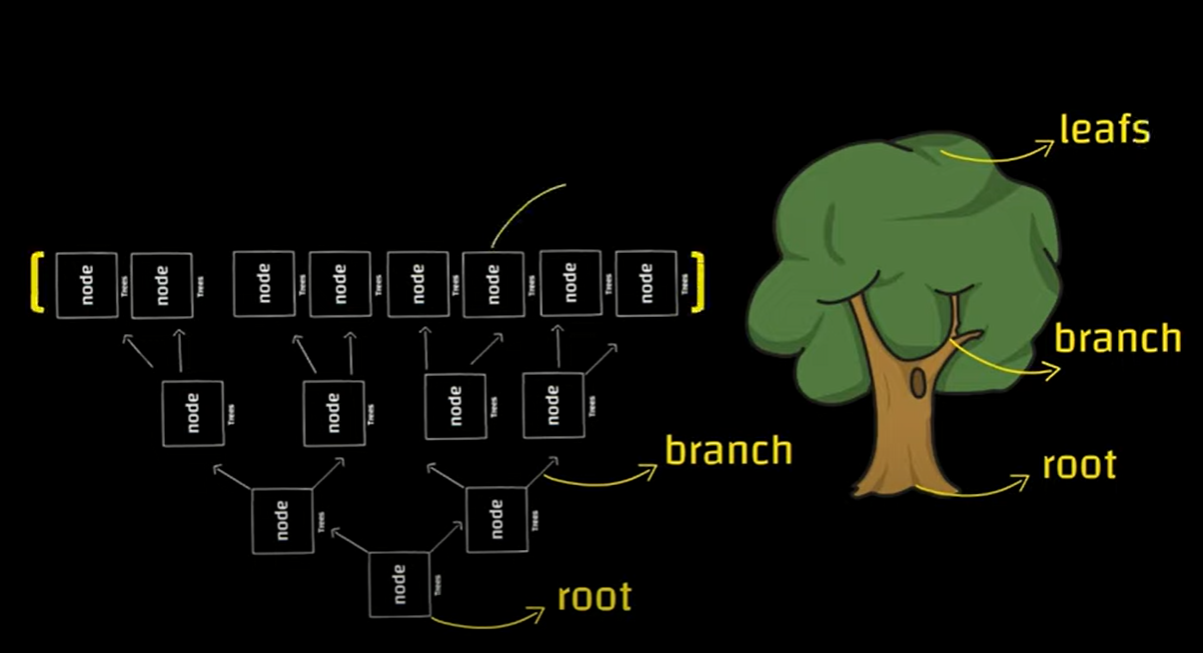
***Know focus you attention here : the impact of this problem appears when we try to search in linked list , and know if i want to search in array i have many searching algorithms like linear search and binary search but in linked i can’t use binary search because i can’t access to middle element , so if i want to search in linked list so i have one algorithm is linear search***

***Make in mind , when you switch between data stracture i will get diffrent features and lose diffrent features , no best data structure , each problem has the best one .***

*And Know i want a data structure that contain linked list features and binary search abbility to applied , this data sturcture is* ***trees***

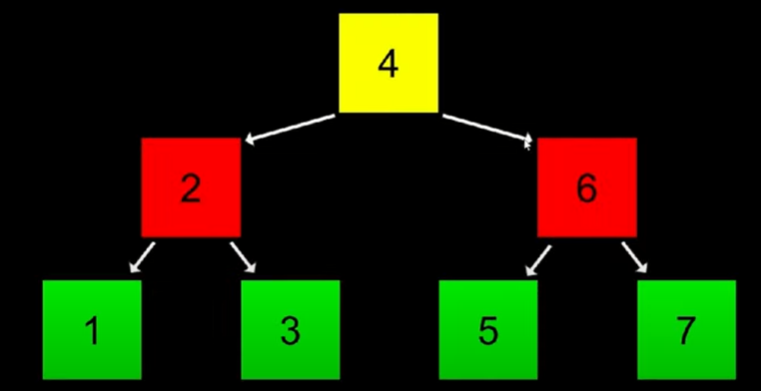
Trees are a little similar to linked list .

Linked list contain node and also trees contain node but diffrence here that in linked list the node point to one other node , but in linked list the pointer point on two nodes As the images show in tree , the node point on two other nodes and each point on two other node . yes this is data structure tree and for prove this to you , the first node in the tree named root and any branch named branch and also the nodes wich anything branch from , named leafs, see the img

we as a humans always we read from the top to the bottom the tree should reed it from the bottom to the top , so the solution is rotate the tree .

**Tree Convert This (Element Orderd By A Linear Manner) :**



To This shap :

I think that something now in you mind when you see the img above , all elements that are in the right of the root all are big from the root and all the elemnt in the left of the root are small from the root , and this also for sub-trees , (node that point two othor node ) , i think this structure remember you in binary search , tree structure make us able to use binary search , and actually this sturcture Its scientific name is **tree binary search** . you remeber that in binary search in array we enter to the middle of the array but here in the tree where is the middle 🡪 Root

**HOW TO MAKE TREE BY CODE**

Know I will store 1 2 3 in a tree ,

Qu : wich number will be the root

***ANSWEAR :*** sure 2 , as we say the root is the middle element , and 1 will be in left and 3 will be in right , because all element right of root are big from it and all elements in root left are small from it .

The first thing is node struct :

 typedef struct node {

         int number ;

         struct node\* left ;

         struct node\* right ;

 }node;

As you see node contain data (number) and two pointers .

Know let’s get started :

1- we will create root element :

NOTE : we must create all nodes in heap section like linked list :

int main (void ){

    node\* tree = NULL ;

    node\* n = malloc(sizeof(node));

    if (n == NULL){

      return 1 ;

    }

    n -> number =  2   ;

    n -> left   = NULL ;

    n -> right  = NULL ;

    tree = n ;

   }

And know we will create all other nodes :

int main (void ){

    node\* tree = NULL ;

    node\* n = malloc(sizeof(node));

    if (n == NULL){

      return 1 ;

    }

    n -> number =  2   ;

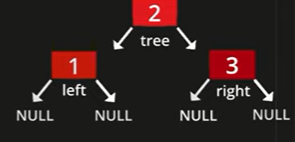
    n -> left   = NULL ;

    n -> right  = NULL ;

    tree = n ;

    n = malloc(sizeof(node));

    if(n == NULL){

          free(tree);

          return 1 ;

    }

     n -> number = 1 ;

     n -> left = NULL ;

     n -> right = NULL ;

     tree -> left = n ;

     n == malloc(sizeof(node));

     if(n == NULL){

       free(tree);

       free(tree -> left);

       return 1 ;

     }

     n -> number = 3 ;

     n -> left = NULL ;

     n -> right = NULL ;

     tree -> right = n ;

   }

***And know I want print all tree element (numbers) :***

So here we have 2 methods : first one is using loop and the second one is creating a function which do that :

1- using loop :

    // this loop made for print root number and all left branch numbers

     for ( node\* tmp = tree ;   tmp != NULL ;   tmp = tmp -> left ){

                    printf("%i ",tmp -> number);

     }

    // this loop made for all right branch numbers

  for ( node\* tmp = tree->right ; tmp != NULL ; tmp = tmp -> right ){

                    printf("%i ",tmp -> number);

     }

2- using function :

 void print\_tree (node\* root){

         if (root == NULL){

          return  ;

         }

         print\_tree(root -> left);

         printf("%i ", root -> number);

         print\_tree(root -> right);

         free(root);

   }

Now we free all element , imagine with me that free line code we writed before print function , so here we will have a big error , because we will free root then we want to print a data that is relate to it so we can’t arrive to it .

And know the os will not free any element before making sure that it’s the last and it will not help us to arrive to any other data .

Meditate the code as well . (meditation)

***Search in binary tree***

*As we know The principale goal or object that push us to discover tree data structure is finding a data structure that contain all linked list features + the abbility for using binary search algorithm , as we know tree data stucture is the best sturcture for this algorithm, the first thing we will create tree node that contain that contain two pointer and data (number).*

**and let’s starting coding :**

So we will create a function that search for an element in a specific tree , so this function will be int type or void type or bool type .

Bool 🡪 true / false .

Int 🡪 0 / 1

Void 🡪 don’t return any value « return ; «

And the function will take :

1- the name of the tree

2- the element i want search for it .

Meditate the code as well . (meditation) .

void search( node\* root , int number ){

        if (root == NULL){

           printf("not found \n");

          return  ;

         }

             else if (root -> number == number){

                  printf("found \n");

                  return ;

             }

             else if(number < root -> number){

                 printf("go left \n");

                 search (root->left , number);

             }

             else if(number > root -> number){

                 printf("go right \n");

                 search (root->right , number);

             }

   }

**All the code :**

#include <stdio.h>

 #include <stdlib.h>

 // 1 2 3

 typedef struct node {

         int number ;

         struct node\* left ;

         struct node\* right ;

 }node;

   void print\_tree (node\* root);

   void search( node\* root , int number );

   int main (void ){

    node\* tree = NULL ;

    node\* n = malloc(sizeof(node));

    if (n == NULL){

      return 1 ;

    }

    n -> number =  2   ;

    n -> left   = NULL ;

    n -> right  = NULL ;

    tree = n ;

    n = malloc(sizeof(node));

    if(n == NULL){

          free(tree);

          return 1 ;

    }

     n -> number = 1 ;

     n -> left = NULL ;

     n -> right = NULL ;

     tree -> left = n ;

     n = malloc(sizeof(node));

     if(n == NULL){

       free(tree);

       free(tree -> left);

       return 1 ;

     }

     n -> number = 3 ;

     n -> left = NULL ;

     n -> right = NULL ;

     tree -> right = n ;

          int num ;

          printf("number : ");

           scanf("%i",&num);

        search(tree , num);

       print\_tree (tree);

   }

   void print\_tree (node\* root){

         if (root == NULL){

          return  ;

         }

         print\_tree(root -> left);

         printf("%i ", root -> number);

         print\_tree(root -> right);

         free(root);

   }

   void search( node\* root , int number ){

        if (root == NULL){

           printf("not found \n");

          return  ;

         }

             else if (root -> number == number){

                  printf("found \n");

             }

             else if(number < root -> number){

                 printf("go left \n");

                 search (root->left , number);

             }

             else if(number > root -> number){

                 printf("go right \n");

                 search (root->right , number);

             }

   }

Know we will speak about tree speed :

1- (log n) steps for search in binary tree and this is a very good speed compared to linked list

2- (log n) step in inserting / linked list (n)

*But for this we must pay a price and this price is MEMORY :*

As we know every node that we create it have to pointe the left and the right and this a very big space especially if you have a big tree (8byte for each pointer) imagine the space if you have a big tree the lose in leafs .

And here you have to choice memory or speed .

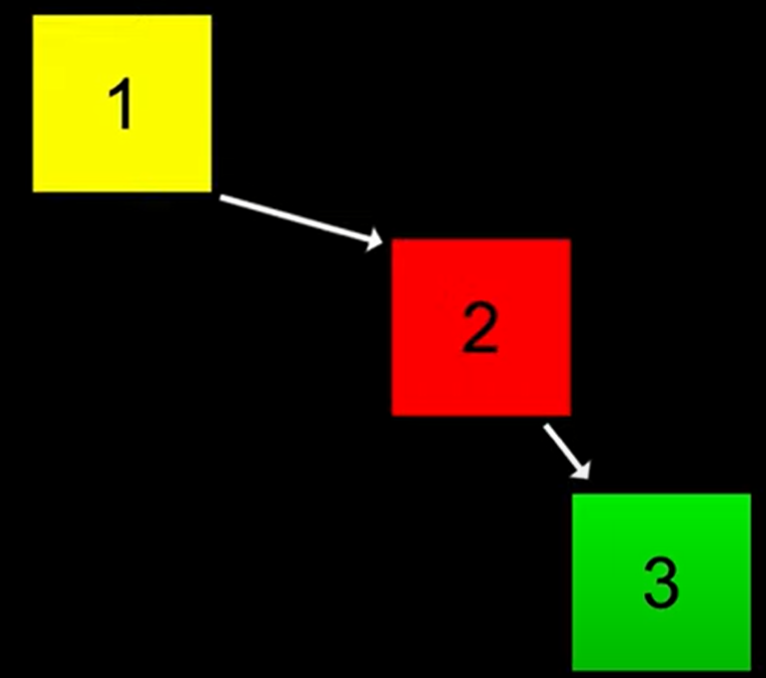
If you want a fast program and you have to memory so choose ***tree***

If you want a porgram which is able to run in a small space and you don’t give a fuck to the speed so choose ***Linked list***

And also we have another problem , sometimes tree become a linked list so you will lose all tree features , for well understanding focus with me here :

When we create a tree and we store in it 1 2 3 , we knew the number , and in the reality we bring these numbers from the user so you don’t know numbers that the user will enter .

Imagine with me that the fisrt number that user enter is 1 and second is 2 and third is 3 , so here as the img show the tree become a linked list :



So here you will ask me about the solution , ANSWEAR : there is many algorithm that makes tree a balanced tree like :

« red black tree »

***Hash Table***

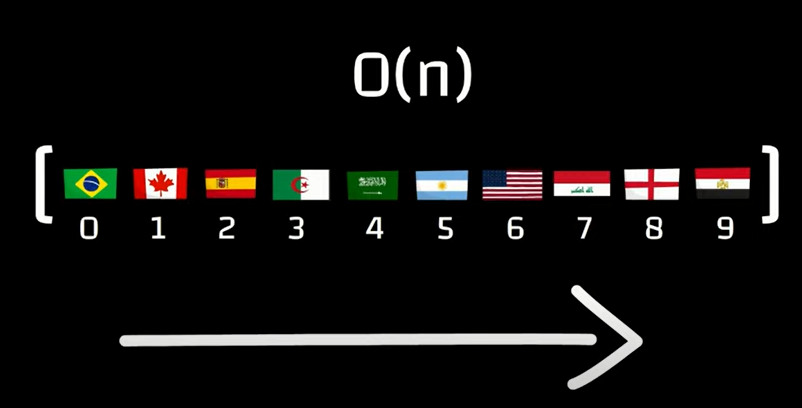
*For well understanding hash table , we have to know, why we learn data structure ? and ANSWEAR : is speed , all data structures store the memory but each one has it own features .*

*Now , in all data types that we learned (linked list – tree) the time taken by searching process is relative to the place of the element , is the element the begging or in the end if the element is in the begging so the speed will be order of 1 o(1), but if the element is at the end so here searching process will take time ,to be honest this time is relative to the size of tree or linked list …*

Simpley linked list will take o(n) , and tree will take o(log n),but me I don’t want o(n) or o(log n) me i want a constant speed o(1) here as we say i don’t mean 1 but i mean constant speed , for well understanding :

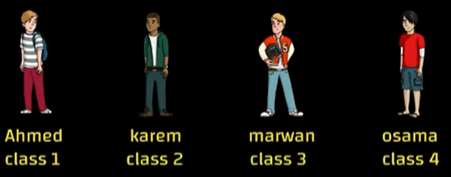
We will take google as an example 🡪 as we know google has millions of pages , and when we search for one of them we get it in a fixed time wich mean it is the same time if i search or another page . and for get constante speed we have to learn hash table .

**So Hash table count on Key and Value , let’s take an example :**

**Imagine with me that you have 10 countries , if we store this countries in a linked list or array searching process will take o(n) by a linnear manner**

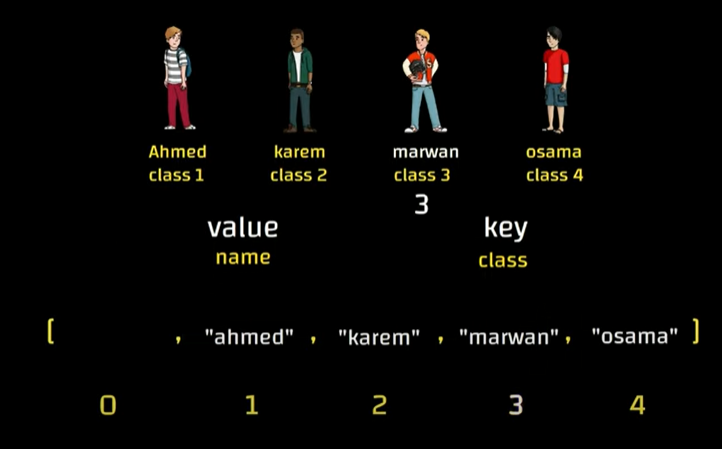
And here you have onther choice is HASH TABLE , as we say hash table count on key and the value , so we class them count on continet , or continet will be the key .



So here searching process will be a little easy , if you want to search for algeria you will not be forced to search in all world countries you will just enter to the key or the hash and search between africa countries , and know we will take an example more realistic , imagine with me that we have 4 students in shchool .

If i want to store this names in an array , but in array i have a problem if i want to search in it i will be forced to search by a linear manner so here the speed will be o(n).

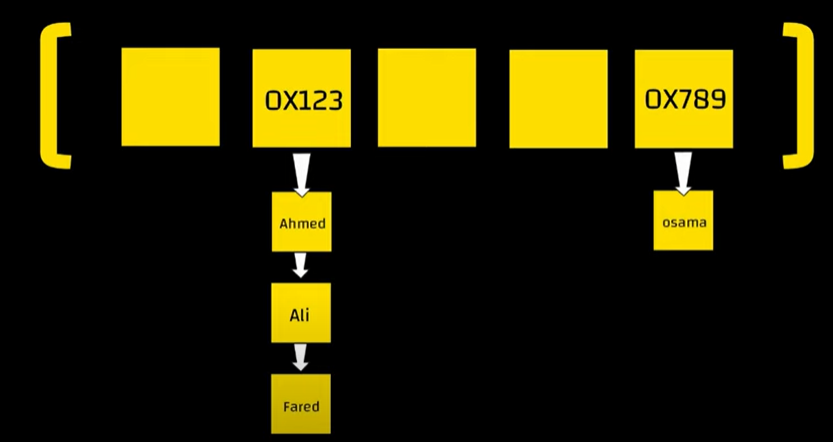
Here hash table comes and say : i will make a key for each value so the key will guide to the value without passing an all element

By the way the key can be all , in this example we will take class number as a key .

So know in array as you know it’s enought to write index number to arrive to the element , so now the speed is o(1) in all cases .

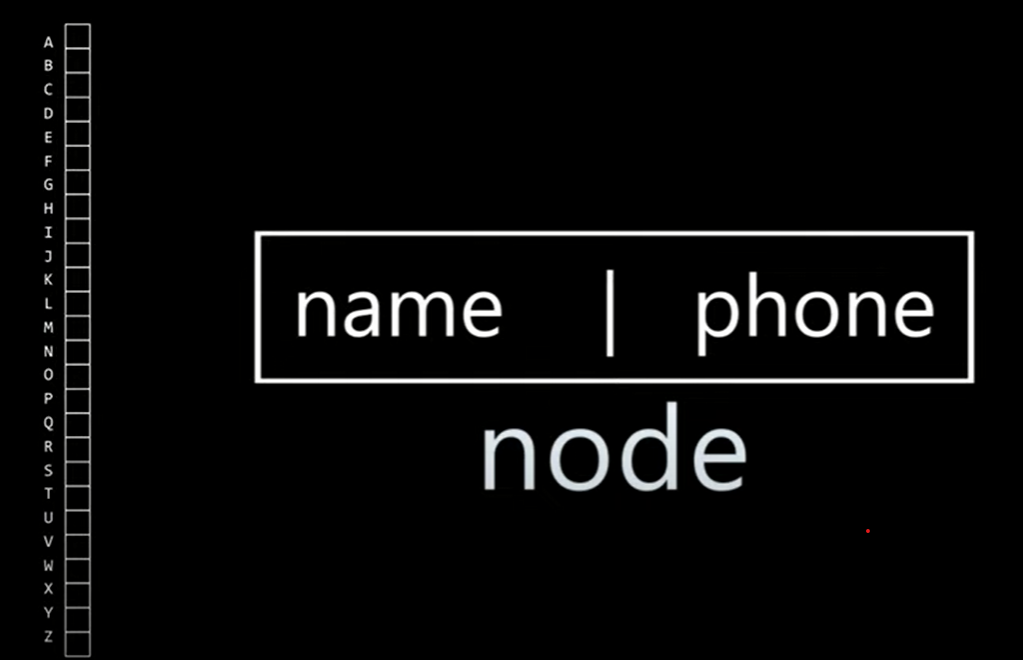
So the pricipale idea is making a key for each value or for each group of values , as we know class can contain more that one students, imagine with me that class 1 contain 3 students so now how can i put them in one index ? each index can’t hold more than one element and if you want to add to it something so the old value will be replaced by the new value .

The solution Here is making linked list tht contain all students names in all the class and the index will be filled by the pointer that point on linked list so each index point on a specific linked list , index 1 contain linked list pointer that contain all sudent class 1 name , vice versa …

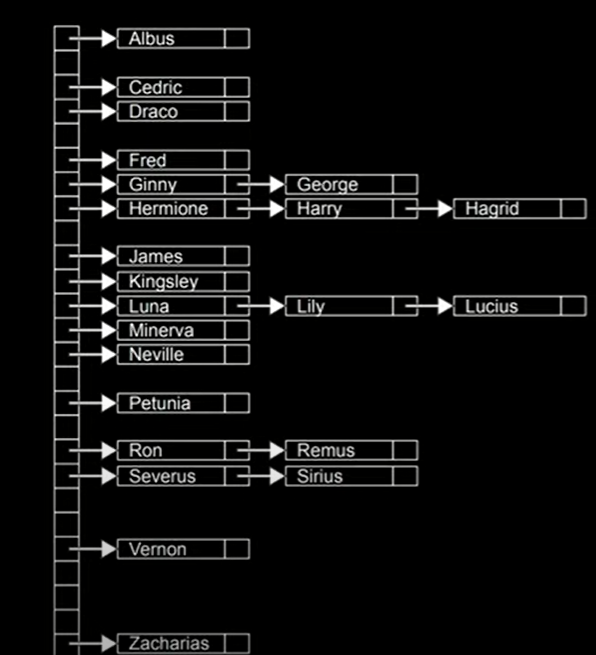
Here you make the problem smaller, but hash table can have a problem , you can have a class that don’t contain any student and onther class contain many students so that will make 2 problems . 1- you reserve a big size in array and you don’t use much element in it and that mean Waste of resources for the void , 2- if you have many student in a class that will cost us many time in searching it relative to linked list size and if it has a big size so here we will lose hash table feature so the speed will be o(n) instead of (1)

**What is the most famous hash table example ?**

***ANSWEAR : Phone Book***

***The name in this phone book will be Alphabetically arranged so know we will create an array that contain 26 element an each index mean a letter . by example index 0 will represent letter A . and each index will have a pointer that refer to linked list that contain all names that start with letter by example by letter A . by the way this linked list will contain nodes that contain name and phone***

So know if you search for Billal you don’t have to search betwenn all name , you have only to enter B linked list and search in it (b linked list pointer you will find it in index [1]

****