DSA assignment

class Node:

    def \_\_init\_\_(self,data):

        self.data=data

        self.next=None

class Stack:

    def \_\_init\_\_(self):

        self.head=None

        self.cur=None

        self.top=None

        self.prev=None

        self.temp2=None

        self.temp=None

        self.temp1=None

        self.temp10=None

        self.temp11=None

        self.temp12=None

        self.temp13=None

        self.temp14=None

        self.temp15=None

        self.temp20=None

    def insert(self,data,stack2):

        newnode=Node(data)

        if(self.cur is None):

            self.insertatEnd(data)

        else:

            if(self.head is None):

                self.head=newnode

                self.temp=self.head

                self.cur=self.head

            else:

                self.temp=self.head

                while(self.temp!=self.cur):

                    self.prev=self.temp

                    self.temp=self.temp.next

                while(self.temp1!=self.cur):

                    self.temp1=self.head

                    while(self.temp1.next is not None):

                        self.temp2=self.temp1

                        self.temp1=self.temp1.next

                    if(self.temp1==self.head):

                        self.head=None

                    else:

                        self.temp2.next=None

                    newnode1=Node(self.temp1.data)

                    if(stack2.head==None):

                        stack2.head=newnode1

                    else:

                        stack2.head.next=newnode1

                if(self.head is None):

                    self.head=newnode

                    self.temp=newnode

                    self.cur=newnode

                else:

                    if(self.cur is not None or self.cur.next is not None):

                        self.prev.next=newnode

                    self.cur=newnode

                    self.temp=newnode

                while(stack2.head is not None):

                    stack2.temp1=stack2.head

                    while(stack2.temp1.next is not None):

                        stack2.temp=stack2.temp1

                        stack2.temp1=stack2.temp1.next

                    if(stack2.temp1==stack2.head):

                        self.temp.next=stack2.head

                        self.temp=self.temp.next

                        stack2.head=None

                    else:

                        stack2.temp.next=None

                        self.temp.next=stack2.temp1

                        self.temp=self.temp.next

                self.temp.next=None

    def insertatEnd(self,data):

        newnode=Node(data)

        if(self.head==None):

            self.head=newnode

        else:

            self.temp10=self.head

            while(self.temp10.next is not None):

                self.temp10=self.temp10.next

            self.temp10.next=newnode

    def printText(self):

        self.temp=self.head

        while(self.temp!=None):

            print(self.temp.data,end=" ")

            self.temp=self.temp.next

        print("")

    def cursorRight(self):

        if(self.cur is None):

            print("not possible")

        elif(self.cur.next==None):

            self.cur=None

        else:

            self.cur=self.cur.next

    def cursorLeft(self):

        if(self.cur==self.head):

            print("not possible")

        else:

            self.temp10=self.head

            if(self.cur is not None):

                while(self.temp10.data!=self.cur.data):

                    self.temp11=self.temp10

                    self.temp10=self.temp10.next

                self.cur=self.temp11

            elif(self.cur is None):

                while(self.temp10.next is not None):

                    self.temp10=self.temp10.next

                self.cur=self.temp10

    def delete(self,stack2):

        self.temp10=self.head

        if(self.cur==None):

            while(self.temp10.next!=None):

                self.temp11=self.temp10

                self.temp10=self.temp10.next

            self.temp11.next=None

        elif(self.head is None):

            print("deletion no possible")

        elif(self.cur==self.head):

            print("deletion not possible")

        else:

            while(self.temp10.next.data!=self.cur.data):

                self.temp11=self.temp10

                self.temp10=self.temp10.next

            self.temp11.next=None

            self.temp12=self.cur

            while(self.temp12 is not None):

                self.temp13=self.temp12

                while(self.temp13.next is not None):

                    self.temp14=self.temp13

                    self.temp13=self.temp13.next

                if(self.temp13==self.temp12):

                    self.temp12=None

                else:

                    self.temp14.next=None

                newnode1=Node(self.temp13.data)

                if(stack2.head is None):

                    stack2.head=newnode1

                    stack2.temp15=stack2.head

                else:

                    stack2.temp15.next=newnode1

            while(stack2.head is not None):

                stack2.temp1=stack2.head

                while(stack2.temp1.next is not None):

                    stack2.temp=stack2.temp1

                    stack2.temp1=stack2.temp1.next

                if(stack2.temp1==stack2.head):

                    self.temp11.next=stack2.head

                    self.temp11=self.temp11.next

                    stack2.head=None

                else:

                    stack2.temp.next=None

                    self.temp11.next=stack2.temp1

                    self.temp11=self.temp11.next

            #please enter code till here only

stack=Stack()

stack2=Stack()

stack.insert(10,stack2)

stack.insert(20,stack2)

stack.insert(30,stack2)

stack.insert(40,stack2)

stack.insert(50,stack2)

stack.printText()

#cursor here is now after 50

stack.cursorRight() # throws an error as we cant go further right

stack.cursorLeft() #takes the cursor to 50

stack.cursorLeft() #takes the cursor to 40

stack.insert(35,stack2) #inserts an element at 40 and 40 50 are popped and pushed again

#the cursor is now at 35

stack.printText()

stack.cursorRight()

stack.cursorRight() #it is now at 50

stack.insert(45,stack2) #inserts 45 by popping out 50 and after insertion it is pushed back

stack.printText()

#now lets delete

#now the cursor is at 45

stack.cursorRight()# now it is at 50 if perform delete it shd remove 45

stack.delete(stack2)

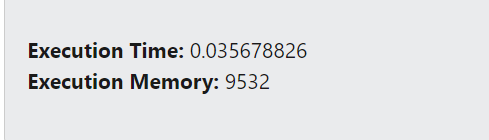
stack.printText()

stack.cursorLeft() # now the cursor is at 50 we sent it 40

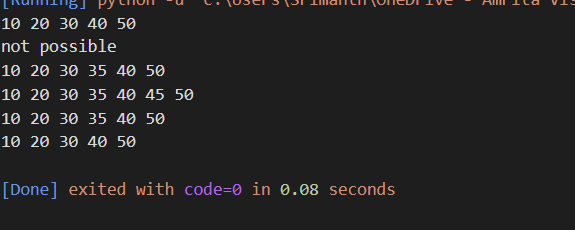
stack.delete(stack2)#it shd remove 35

stack.printText()

Execution time and memory according to hpoj



Acc to vscode



Now lets do it arrays

class Stack:

    def \_\_init\_\_(self):

        self.stack=[]

        self.temp=[]

        self.cursor=0

        self.cursor1=self.cursor-1

    def insertatcursor(self,data):

        if(len(self.stack)==0):

            self.stack.append(data)

            self.cursor+=1

        elif(self.cursor==len(self.stack)):

            self.stack.append(data)

            self.cursor+=1

        else:

            i=len(self.stack)

            while(i>self.cursor):

                element=self.stack[len(self.stack)-1]

                self.stack.pop()

                self.temp.append(element)

                i=i-1

            self.stack.append(data)

            while(len(self.temp)!=0):

                element1=self.temp[len(self.temp)-1]

                element1=self.temp.pop()

                self.stack.append(element1)

    def cursorright(self):

        if(self.cursor==len(self.stack)):

            print("cant move the cursor")

        else:

            self.cursor=self.cursor+1

    def cursorleft(self):

        if(self.cursor==0):

            print("cant move the cursor")

        else:

            self.cursor=self.cursor-1

    def getText(self):

        for i in self.stack:

            print(i,end=" ")

        print("")

    def delete(self):

        if(self.cursor==0):

            print("deletion not possible")

        else:

            i=len(self.stack)-1

            while(i>=self.cursor):

                element=self.stack[len(self.stack)-1]

                self.stack.pop()

                self.temp.append(element)

                i=i-1

            self.stack.pop()

            while(len(self.temp)!=0):

                element1=self.temp[len(self.temp)-1]

                element1=self.temp.pop()

                self.stack.append(element1)

            self.cursor=self.cursor-1

    def printcursor(self):

        print(self.cursor)

stack=Stack()

stack=Stack()

stack.insertatcursor(10)

stack.insertatcursor(20)

stack.insertatcursor(30)

stack.insertatcursor(40)

stack.insertatcursor(50)

stack.getText()

#cursor here is now after 50

stack.cursorright() # throws an error as we cant go further right

stack.cursorleft() #takes the cursor to 50

stack.cursorleft() #takes the cursor to 40

stack.insertatcursor(35) #inserts an element at 40 and 40 50 are popped and pushed again

#the cursor is now at 35

stack.getText()

stack.cursorright()

stack.cursorright() #it is now at 50

stack.insertatcursor(45) #inserts 45 by popping out 50 and after insertion it is pushed back

stack.getText()

#now lets delete

#now the cursor is at 45

stack.cursorright()# now it is at 50 if perform delete it shd remove 45

stack.delete()

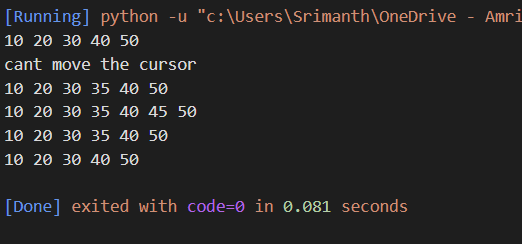
stack.getText()

stack.cursorleft() # now the cursor is at 50 we sent it 40

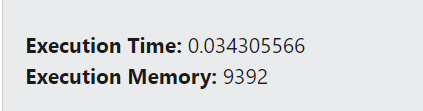
stack.delete()#it shd remove 35

stack.getText()

acc to vscode the execution time



Acc to hpoj



Time complexity:

As we have seen stack using linked list and arrays almost took the same time the difference between them is negligible

But due to the excessive no of loops used to maintain pointers we can say that linked list is a bit faster than arrays

Space complexity:

As we can see Arrays took lesser space compared to time because the remaining memory is taken for storing temporary address

And even in the code many pointers were used to reduce the complexity of the code but time complexity increased due to that.

So due to the extra pointers used the memory usage is more in linked list

Code complexity:

Array implementation was very simple even the number of lines are very less

But linked list code was very complex as we have maintain many pointers etc which increases the complexity of code

Insertions and deletions:

Almost the same thing as insertion in between and at the end takes O(n) time in linked list

And insertion at the start and middle takes O(n) time is arrays they are almost the same

But if a tail pointer is used we can decrease the time complexity for insertion at the end

So if a tail pointer is used then linked list overtakes arrays in this case

Random access:

It is not possible in linked list but is in arrays. But stack is a data structure in which there is no scope for random access so any data structure does not take a step ahead here.

But if we need an application where we need to use a stack and return the elements randomly the arrays are helpful

Memory usage:

Linked list use more memory than arrays

But linked list is more memory efficient as it uses all the memory and doenot waste memory . it doesnot need contiguous memory blocks.

But if we declare a size of array 100 and we just push 4 elements we are wasting a lot of memory allocated and array needs contiguous memory locations as well.

So for large applications we need to store data in huge amounts then linked list almost consumes double the memory compared to arrays . so for applications with small amount of data we can use linked list as it is faster but for huge applications better to use arrays.