Basic needs for Odoo

Linked List vs Array:

تفاوت های ارایه و لینک لیست:
برای حذف و اضافه کردن به لینک لیست ها راحت تر از ارایه ها هست، چون کافیه فقط ادرس اشاره گر به بعدی را عوض کنیم و ان ایتم از لینک لیست حذف میشود و ان مقدار چون به هیچ حایی رفرنس ندارد به طور خودکار وقتی استک خودش را تمیز میکند پاک میشود.
ارایه ها فضای ذخیره بیشتری از لینک لیست ها را اشغال میکنند.
در ارایه ها شما می‌توانید صورت رندوم دسترسی داشته باشید. و استفاده انها راحت تر است.

Stack and Queue Heap:

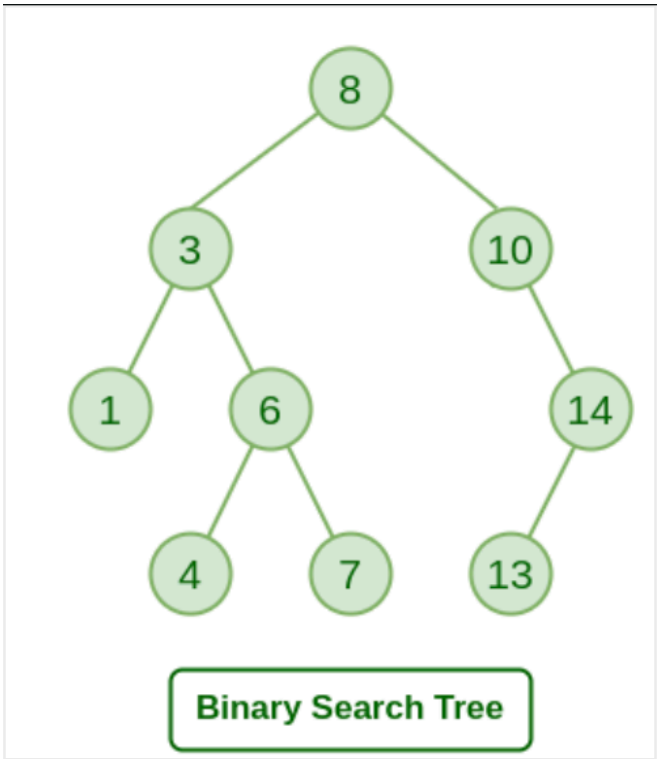
استک در واقع یک نوع ساختار برای ذخیره سازی دیتا تایپ ها هستند؛ که در استک از ترتیب **FILO یا LIFO استفاده میشود.(First In Last Out)(Last In First Out). در واقع در FILO به این صورت است که اولین مقداری که وارد میشود اخرین مقداری است که خارج میشود و در LIFO به این صورت است که اخرین مقداری که وارد میشود اولین مقداری است که خارج میشود.**
در queue که نوع ساختاری ذخیره سازی است مثل استک به این صورت است که از الگوریتم مرتب سازی FIFO استفاده میکند؛ یعنی اولین عنصری که اضافه میشود اولین عنصری است که خارج میشود. از deque هم استفاده میشود تا:خود queue چاپ شود. و از popleft استفاده می‌شود تا اولین عنصر به طور سریع تر پاک شود.

```
from collections import deque
queue = deque(["Ram", "Tarun", "Asif", "John"])
print(queue)
queue.append("Akbar")
print(queue)
queue.append("Birbal")
print(queue)
print(queue.popleft())
print(queue.popleft())
print(queue)

result:
deque(['Ram', 'Tarun', 'Asif', 'John'])
deque(['Ram', 'Tarun', 'Asif', 'John', 'Akbar'])
deque(['Ram', 'Tarun', 'Asif', 'John', 'Akbar', 'Birbal'])
Ram
Tarun
deque(['Asif', 'John', 'Akbar', 'Birbal'])
```

Binary Search Tree:

یک نوع ساختمان ذخیره دیتا است که به صورت درخت شکل است. که هر نود حداکثر دو تا فرزند دارد که فرزند سمت چپی مقدار کمتری از والد خود دارد اما مقدار سمت راستی بیشتر از والد خود است. این الگوریتم برای حذف اضافه و سرچ کردن داخل دیتا ها راحت تر و سریع تر است.



recursion:

تابع بازگشتی، تابعی است که مدام خودش را صدا میزند تا به نقطه مورد نظر برسد البته نکته ای که وجود دارد باید یک نقطه پایان هم برایش تغیین کنیم که به فلان نقطه رسید متوقف شود وگرنه همینجور ادامه میدهد تا برنامه کرش کند.

```
1 def Factorial(number):
2   if (number == 1):
3     return 1;
4   return number * Factorial(number - 1);
5
6 print(Factorial(5));
```

```
120
```

Sorting Algorithms:

بحث الگوریتم های مرتب سازی رو اینگونه شروع کنیم که سه نوع الگوریتم مرتب سازی داریم که جلو تر به انها میرسیم که از دو نوع Increasing Order , Decreasing Order پیروی میکنند که هر کدام از این دو یک نوع زیرمجموعه خود دارند که Non-Increasing Order و Non-Decreasing Order هستند. نوع Decreasing Order یعنی اعداد به صورت نزولی مرتب شده اند(از بزرگ به کوچک) و نوع Increasing Order یعنی اعداد به صورت صعودی مرتب شده اند(کوچک به بزرگ). و در دو مورد دیگر همونگونه اند با این تفاوت که عصنر i ام مقدارش به i-1 برابر یا کوچک تر باشد و در دومی هم عنصر i ام مقدارش با i-1 برابر یا بزرگ تر باشد.

الگوریتم ها:

- Bubble Sort
- Selection Sort
- Insertion Sort

در Bubble Sort در هر دوره دو عنصر را قیاس میکند و اگر کوچیک تر از عنصر قبلی باشد انها را جا به جا میکند و همینجور ادامه میدهد تا یک دوره تمام شود و اگر همچنان اعداد مرتب نبودند یک دوره جدید رو شروع میکند.

Sequence: 2, 23, 10, 1

First Iteration

*(**2, 23**, 10, 1) –> (**2, 23**, 10, 1), Here the first 2 elements are compared and remain the same because they are already in ascending order.*

*(2, **23, 10**, 1) –> (2, **10, 23**, 1), Here 2nd and 3rd elements are compared and swapped(10 is less than 23) according to ascending order.*

*(2, 10, **23, 1**) –> (2, 10, **1, 23**), Here 3rd and 4th elements are compared and swapped(1 is less than 23) according to ascending order*

At the end of the first iteration, the largest element is at the rightmost position which is sorted correctly.

Second Iteration

*(**2, 10**, 1, 23) –> (**2, 10**, 1, 23), Here again, the first 2 elements are compared and remain the same because they are already in ascending order.*

*(2, **10, 1**, 23) –> (2, **1, 10**, 23), Here 2nd and 3rd elements are compared and swapped(1 is less than 10) in ascending order.*

At the end of the second iteration, the second largest element is at the adjacent position to the largest element.

Third Iteration

*(**2, 1**, 10, 23) –> (**1, 2**, 10, 23), Here the first 2 elements are compared and swap according to ascending order.*

The remaining elements are already sorted in the first and second Iterations. After the three iterations, the given array is sorted in ascending order. So the final result is 1, 2, 10, 23.

```
# Python3 program for Bubble Sort Algorithm Implementation
def bubbleSort(arr):

    n = len(arr)

    # For loop to traverse through all
    # element in an array
    for i in range(n):
        for j in range(0, n - i - 1):

            # Range of the array is from 0 to n-i-1
            # Swap the elements if the element found
            #is greater than the adjacent element
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

# Driver code

# Example to test the above code
arr = [ 2, 1, 10, 23 ]

bubbleSort(arr)

print("Sorted array is:")
for i in range(len(arr)):
    print("%d" % arr[i])
```

در Selection Sort به اینگونه است که در هر دوره میگردد به دنبال کوچک ترین عنصر و ان را تا قبل از عنصر های بزرگتر از ان میگذارد.

Sequence: 7, 2, 1, 6

*(7, 2, **1**, 6) –> (**1**, 7, 2, 6), In the first traverse it finds the minimum element(i.e., 1) and it is placed at 1st position.*

*(1, 7, **2**, 6) –> (1, **2**, 7, 6), In the second traverse it finds the 2nd minimum element(i.e., 2) and it is placed at 2nd position.*

*(1, 2, **7**, **6**) –> (1, 2, **6**, **7**), In the third traverse it finds the next minimum element(i.e., 6) and it is placed at 3rd position.*

After the above iterations, the final array is in sorted order, i.e., 1, 2, 6, 7.

```
# Selection Sort algorithm in Python
def selectionSort(array, size):

    for s in range(size):
        min_idx = s

        for i in range(s + 1, size):

            # For sorting in descending order
            # for minimum element in each loop
            if array[i] < array[min_idx]:
                min_idx = i

        # Arranging min at the correct position
        (array[s], array[min_idx]) = (array[min_idx], array[s])

# Driver code
data = [ 7, 2, 1, 6 ]
size = len(data)
selectionSort(data, size)

print('Sorted Array in Ascending Order is :')
print(data)
```

در Insertion Sort دو عنصر انتخاب میشوند از اول لیست و با یکدیگر قیاس می‌شوند و و مرتب سازی انجام میگیردبا این تقاوت که وقتی دو عنصر جا به جا شدند اگر همچنان عنثر جا به جا شده کوچک تر از عنصر قبلی خود باشد جا به جا میشود.

Sequence: 7, 2, 1, 6

*(**7**, **2**, 1, 6) –> (**2**, **7**, 1, 6), In the first iteration, the first 2 elements are compared, here 2 is less than 7 so insert 2 before 7.*

*(2, **7**, **1**, 6) –> (2, **1**, **7**, 6), In the second iteration the 2nd and 3rd elements are compared, here 1 is less than 7 so insert 1 before 7.*

*(**2**, **1**, 7, 6) –> (**1**, **2**, 7, 6), After the second iteration (1, 7) elements are not in ascending order so first these two elements are arranged. So, insert 1 before 2.*

(1, 2, 7, 6) –> (1, 2, 6, 7), During this iteration the last 2 elements are compared and swapped after all the previous elements are swapped.

```
# Creating a function for insertion sort algorithm
def insertion_sort(list1):

        # Outer loop to traverse on len(list1)
        for i in range(1, len(list1)):

            a = list1[i]

            # Move elements of list1[0 to i-1],
            # which are greater to one position
            # ahead of their current position
            j = i - 1

            while j >= 0 and a < list1[j]:
                list1[j + 1] = list1[j]
                j -= 1

            list1[j + 1] = a

        return list1

# Driver code
list1 = [ 7, 2, 1, 6 ]
print("The unsorted list is:", list1)
print("The sorted new list is:", insertion_sort(list1))
```

اگر ما بخوایم قبل و بعد از عمل اصلی خود یکسری کار انجام بشود. از این استفاده میکنیم. که با دو کلمه with و open استفاده میشود. یک کلاس مینویسیم برای عمل هایی که میخواهیم قبل و بعد اتفاقا بیافتد.
یکی از کاربرد ها:

```
with open('message.txt') as f:
   pass
```

برای ساخت context manager باید دو متد __enter__ و __exit__ را داشته باشد.

```
class A:
   def __enter__(self):
      print('Entering context manager')
```

```
   def __exit__(self, exc_type, exc_val, exc_tb):
      print('Exiting context manager')
```

برای فعال کردن ان از with استفاده میکنیم.

```
with A():
   show()

output:
Entering context manager
Show Function...
Exiting context manager
```