Data and File Structures Laboratory

Data Structures with Python

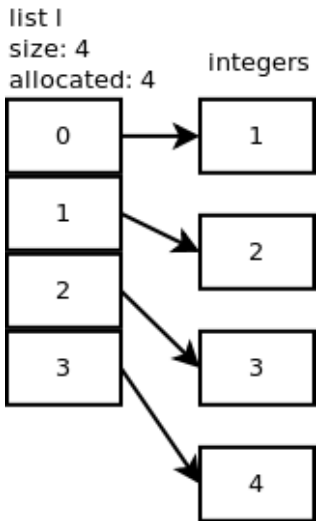
Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
Indian Statistical Institute, Kolkata

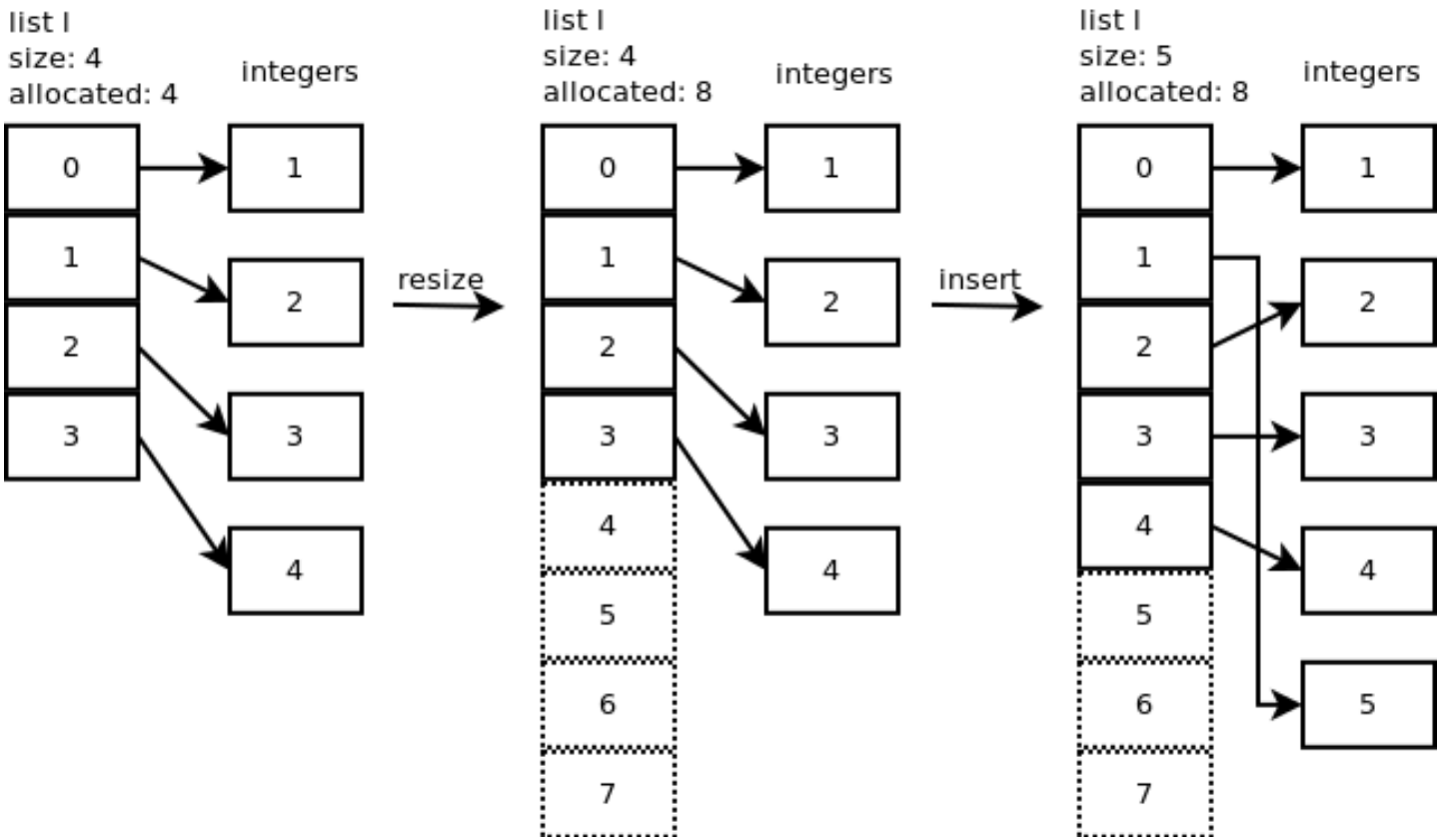
November, 2019

Lists



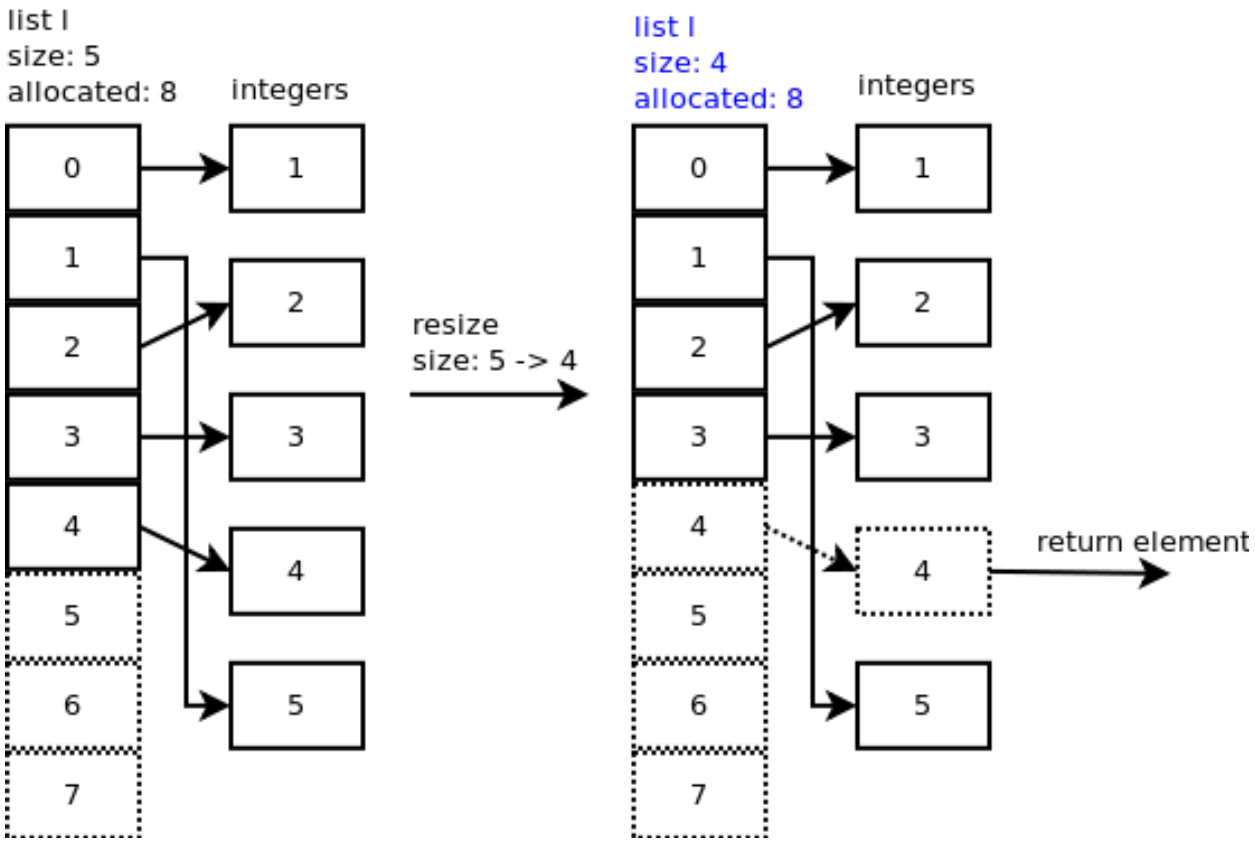
- *Length* of the list $= n$.
- n *linked* memory locations that gets dynamically reallocated.
- The elements are homogeneous but the type is generic.
- *Elements* can be mapped to each of the n memory locations.
- Elements are indexed 0 through $n-1$ ($\equiv -n$ through -1).

Lists – Insert




```
List = []
List.append(<data>) # Inserts at the end
List.insert(<index>, <data>) # Inserts at the <index>
```

Lists – Pop




```
List = []
<data> = List.pop()
```

Initialization

Considering the stack size as static:

```
StackDS = []
SizeStackDS = 10
```

Push operation

```
def StackPUSH(Data, StackDS):
    if len(StackDS) < SizeStackDS:
        StackDS.append(Data)
    else:
        print("Stack overflow!!!")
```

Pop operation

```
def StackPOP(StackDS):
    if len(StackDS) > 0:
        StackDS.pop()
    else:
        print("Stack underflow!!!")
```

Note: If you wish to receive the popped element, include a return statement.

Displaying the elements

```
def StackDISPLAY(StackDS):
    print("The elements in the stack are: ")
    for Element in StackDS:
        print(Element)
```

Note: You can directly write `print(StackDS)`.

Initialization

Considering the queue size as static:

```
QueueDS = []
SizeQueueDS = 10
```

Insert operation

```
def QueueINSERT(Data, QueueDS):
    if len(QueueDS) < SizeQueueDS:
        QueueDS.insert(0, Data)
    else:
        print("Queue overflow!!!")
```

Delete operation

```
def QueueDELETE(QueueDS):
    if len(QueueDS) > 0:
        QueueDS.pop()
    else:
        print("Queue underflow!!!")
```

Note: If you wish to receive the deleted element, include a return statement.

Displaying the elements

```
def QueueDISPLAY(QueueDS):
    print("The elements in the queue are: ")
    for Element in QueueDS:
        print(Element)
```

Note: You can directly write `print(QueusDS)`.

Depth-First Search

The DFS algorithm works as follows:

1. Keep any one of the graph's vertices on top of a stack.
2. Pop out the top data item from the stack and add it to the `Visited` list.
3. Create a list of that vertex's adjacent nodes. Add the ones which are not in the `Visited` list to the top of stack.
4. Repeat steps 2-3 until the stack is empty.

Depth-First Search

```
# Adjacency list defined as a dictionary
Graph = {
    '0' : ['1','2'], '1' : ['3', '4'], '2' : ['5'],
    '3' : [], '4' : ['5'], '5' : []
}
Visited = [] # Array to keep track of visited vertices
def DFS(Visited, Graph, Vertex):
    if Vertex not in Visited:
        Visited.append(Vertex)
        print(Visited[len(Visited) - 1]) # Top of stack
        for Adjacent in Graph[Vertex]:
            DFS(Visited, Graph, Adjacent)
DFS(Visited, Graph, '0') # Function call with vertex '0'
```

Breadth-First Search

The DFS algorithm works as follows:

1. Insert any one of the graph's vertices in the front of a queue.
2. Delete the rear data item from the queue and add it to the `Visited` list.
3. Create a list of that vertex's adjacent nodes. Add the ones which are not in the `Visited` list to the front of the queue.
4. Repeat steps 2-3 until the stack is empty.

Problems – Day 26

1. Write a program to implement a priority queue. While constructing and initializing the queue, ask from the user whether it will be a maximum or minimum priority queue.
2. Write a program to implement a doubly linked list. Note that, a doubly linked list is a type of linked list in which each node has a data item and a pair of links. The first link points to the previous node in the list and the second link points to the next node in the list.
3. Write a program to implement a height balanced tree that has the maximum balance factor of 2.

Problems – Day 26

4. Write a program to implement breadth-first search.
5. Write a program to find out the largest element in a Λ-bitonic matrix provided as user input in a separate file.
6. Write a program to find the frequencies of all the words present in a file, which is provided as user input. Assume that the file contains only alphanumeric characters.