Arrays

Arrays [1]

- The array as an abstract data type
- Class as structure
- **The polynomial Abstract Data Type**
- **The Sparse Matrix Abstract Data Type**
- **The Representation of Multidimensional Arrays**

Arrays Ordered List (1)

- Linear List or Ordered List
- Example
  - (item1, item2, item3, …, item$n$)
  - (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday)
  - (Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King)
  - (basement, lobby, mezzanine, first, second)
  - (1941, 1942, 1943, 1944, 1945)
  - $(a_1, a_2, a_3, \ldots, a_{n-1}, a_n)$

Arrays Ordered List (2)

- The Ordered List Abstract Data Type
  - OrderedList()
    - creates a new ordered list that is empty.
    - It needs no parameters and returns an empty list.
  - add(item)
    - adds a new item to the list making sure that the order is preserved.
    - It needs the item and returns nothing.
    - Assume the item is not already in the list.
  - remove(item)
    - removes the item from the list.
    - It needs the item and modifies the list.
    - Assume the item is present in the list.
  - search(item)
    - searches for the item in the list.
    - It needs the item and returns a boolean value.
  - is_empty()
    - tests to see whether the list is empty.
    - It needs no parameters and returns a boolean value.
  - size()
    - returns the number of items in the list.
    - It needs no parameters and returns an integer.
  - index(item)
    - returns the position of item in the list.
    - It needs the item and returns the index or None.
    - Assume the item is in the list.

Arrays Ordered List (3-lab-ans-1)

- Lab: ex00.py
  - Implements the ordered list ADT
  - OrderedList()
  - add(item)
  - remove(item)
  - search(item)
  - is_empty()
  - size()

```
*data, = 53, 17, 34, 23, 15, 43
print(data)

o = OrderedList()
print(o.is_empty())

for i in data:
    o.add(i)

print(o.is_empty())
print(o)
o.remove(23)
print(o)

print(o.search(43))
print(o.index(23))
```

- ex00.py

```
class OrderedList:
    def __init__(self):
        self.elems = []

    def is_empty(self):
        return not bool(self.elems)

    def add(self, elem):
        if not self.elems:
            self.elems.append(elem)
            return
        cur = 0
        while cur < len(self) and self[cur] <= elem:
            cur += 1

        self.elems.insert(cur, elem)

    def remove(self, elem):
        self.elems.remove(elem)

    def search(self, elem):
        cur = 0
        while cur < len(self) and self[cur] != elem:
            cur += 1
        return False if cur >= len(self) else True
```

Arrays Ordered List (3-lab-ans-2)

- ex00.py

```
    def __len__(self):
        return len(self.elems)

    def __getitem__(self, index):
        return self.elems[index]

    def __str__(self):
        return str(self.elems)
```

Arrays Sparse Matrix (1)

- Sparse Matrix
  - In mathematics, a matrix contains m rows and n columns of elements,
  - we write m X n to designate a matrix with m rows and n columns

Dense Matrix 5 * 3 6 * 6 **Original Matrix**

	col 0	col 1	col 2
row 0	−27	3	4
row 1	6	82	−2
row 2	109	−64	11
row 3	12	8	9
row 4	48	27	47

(a) 15/15

	col 0	col 1	col 2	col 3	col 4	col 5
row 0	15	0	0	22	0	−15
row 1	0	11	3	0	0	0
row 2	0	0	0	−6	0	0
row 3	0	0	0	0	0	0
row 4	91	0	0	0	0	0
row 5	0	0	28	0	0	0

(b) 8/36

Figure 2.3: Two matrices

Arrays Sparse Matrix (2)

- The standard representation of a matrix is a two dimensional array defined as
  - a[MAX_ROWS][MAX_COLS]
  - a[m][n]
- We can locate quickly any element by writing a[i][j]

- Sparse Matrix
  - A matrix is called sparse if it consists of many zero entries.
- Sparse matrix wastes space
  - Implementing a spare matrix by a two-dimensional array waste a lot of memory.
  - we must consider alternate forms of representation.

Arrays Sparse Matrix (3)

- Sparse Matrix Representation
  - use the triple <row, col, value> to represent an element.
  - store the triples by rows.
  - for each row, the column indices are in ascending order.
  - store the number of rows, columns, and nonzero elements.

	col 0	col 1	col 2	col 3	col 4	col 5
row 0	15	0	0	22	0	−15
row 1	0	11	3	0	0	0
row 2	0	0	0	−6	0	0
row 3	0	0	0	0	0	0
row 4	91	0	0	0	0	0
row 5	0	0	28	0	0	0

		row	col	value
smArray	[0]	0	0	15
	[1]	0	3	22
	[2]	0	5	-15
	[3]	1	1	11
	[4]	1	2	3
	[5]	2	3	-6
	[6]	4	0	91
	[7]	5	2	28

Arrays Sparse Matrix (4-lab)

- Lab: matrix_sparse.py
  - use class term as the triple <row, col, value> to represent an element.
  - Implements the method “build_matrix_sparse” with 2d-mat

```
data = [
    [15, 0, 0, 22, 0, -15],
    [0, 11, 3, 0, 0, 0],
    [0, 0, 0, -6, 0, 0],
    [0, 0, 0, 0, 0, 0],
    [91, 0, 0, 0, 0, 0],
    [0, 0, 28, 0, 0, 0],
]

print("sparse matrix >>")
mat = MatrixSparse()
mat.build_matrix_sparse(data)
print(mat)
```

```
sparse matrix >>
(0, 0, 15)
(0, 3, 22)
(0, 5, -15)
(1, 1, 11)
(1, 2, 3)
(2, 3, -6)
(4, 0, 91)
(5, 2, 28)
```

- Class Term

```
class Term:
    def __init__(self, row=0, col=0, value=0):
        self.row = row
        self.col = col
        self.value = value

    def __str__(self):
        return f"{self.row, self.col, self.value}"

    def __repr__(self):
        return str(self)


class MatrixSparse:
    def __init__(
        self, rows=0, cols=0, size=0, sparse = None
    ):
        self.rows = rows
        self.cols = cols
        self.size = size
        self.sparse = sparse

    def build_matrix_sparse(self, mat):
        …
```

Arrays Sparse Matrix (4-lab-ans)

- Lab: matrix_sparse.py
  - use class term as the triple <row, col, value> to represent an element.
  - Implements the method “build_matrix_sparse” with 2d-mat

```
data = [
    [15, 0, 0, 22, 0, -15],
    [0, 11, 3, 0, 0, 0],
    [0, 0, 0, -6, 0, 0],
    [0, 0, 0, 0, 0, 0],
    [91, 0, 0, 0, 0, 0],
    [0, 0, 28, 0, 0, 0],
]

print("sparse matrix >>")
mat = MatrixSparse()
mat.build_matrix_sparse(data)
print(mat)
```

```
sparse matrix >>
(0, 0, 15)
(0, 3, 22)
(0, 5, -15)
(1, 1, 11)
(1, 2, 3)
(2, 3, -6)
(4, 0, 91)
(5, 2, 28)
```

- Class Term

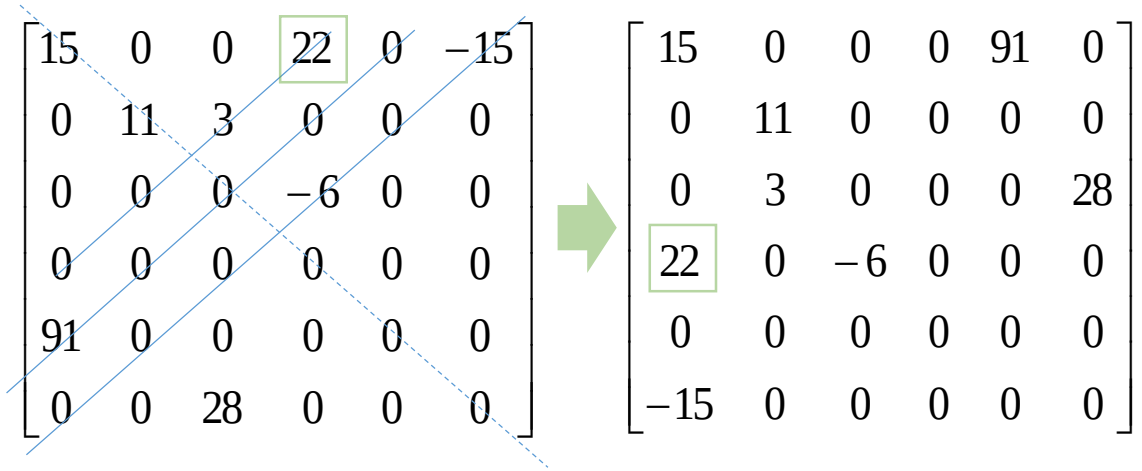
```
def build_matrix_sparse(self, mat):
        self.rows = len(mat)
        self.cols = len(mat[0])

        self.sparse = [
            Term(r, c, v)
            for r, row in enumerate(mat)
            for c, v in enumerate(row)
            if v != 0
        ]
        self.size = len(self.sparse)
```

Arrays Sparse Matrix (5-lab-ans)

- Transposing a matrix

$$(A^T)_{ij} = A_{ji}$$



- Lab: transpose.py

```
data = [
    [15, 0, 0, 22, 0, -15],
    [0, 11, 3, 0, 0, 0],
    [0, 0, 0, -6, 0, 0],
    [0, 0, 0, 0, 0, 0],
    [91, 0, 0, 0, 0, 0],
    [0, 0, 28, 0, 0, 0],
]

print(“transpose matrix >>")
transpose_mat(data)
```

- transepose.py

```
def transpose_mat(mat):
    rows = len(mat)
    cols = len(mat[0])

    ret_mat = [[0] * rows for _ in range(cols)]

    for row in range(rows):
        for col in range(cols):
            ret_mat[col][row] = mat[row][col]

    return ret_mat
```

Arrays Sparse Matrix (6)

- On sparse matrix representation
  - Transpose

- n element at [i][j] will be at [j][i] for (each row i)
  take element (i, j, value) and store it in (j, i, value) of the transpose;
- difficulty: where to put <j, i, value>
  - (0, 0, 15) → (0, 0, 15)
  - (0, 3, 22) → (3, 0, 22)
  - (0, 5, -15) → (5, 0, -15)
  - (1, 1, 11) → (1, 1, 11)
  - need to insert many new triples, elements are moved down very ofte
- Find the elements in the order
  - for (all elements in column j)
    place element (i, j, value) in position (j, i, value);

	row	col	value
smArray [0]	0	0	15
[1]	0	3	22
[2]	0	5	-15
[3]	1	1	11
[4]	1	2	3
[5]	2	3	-6
[6]	4	0	91
[7]	5	2	28

	row	col	value
smArray [0]	0	0	15
[1]	3	0	22
[2]	5	0	-15
[3]	1	1	11
[4]	2	1	3
[5]	3	2	-6
[6]	0	4	91
[7]	2	5	28

	row	col	value
smArray [0]	0	0	15
[1]	0	3	22
[2]	0	5	-15
[3]	1	1	11
[4]	1	2	3
[5]	2	3	-6
[6]	4	0	91
[7]	5	2	28

	row	col	value
smArray [0]	0	0	15
[1]	0	4	91
[2]	1	1	11
[3]	2	1	3
[4]	2	5	28
[5]	3	0	22
[6]	3	2	-6
[7]	5	0	-15

Arrays Sparse Matrix (7-lab-ans)

- Lab: matrix_spase.py
  - Sparse Matrix (4-lab)

```
data = [
    [15, 0, 0, 22, 0, -15],
    [0, 11, 3, 0, 0, 0],
    [0, 0, 0, -6, 0, 0],
    [0, 0, 0, 0, 0, 0],
    [91, 0, 0, 0, 0, 0],
    [0, 0, 28, 0, 0, 0],
]

print("sparse matrix >>")
mat = MatrixSparse()
mat.build_matrix_sparse(data)
print(mat)

print("transpose >>")
mat = mat.transpose()
print(mat)
```

```
sparse matrix >>
(0, 0, 15)
(0, 3, 22)
(0, 5, -15)
(1, 1, 11)
(1, 2, 3)
(2, 3, -6)
(4, 0, 91)
(5, 2, 28)
```

```
transpose >>
(0, 0, 15)
(0, 4, 91)
(1, 1, 11)
(2, 1, 3)
(2, 5, 28)
(3, 0, 22)
(3, 2, -6)
(5, 0, -15)
```

- matrix_sparse.py

```
def transpose(self):
        if self.sparse is None:
            return

        sparse= [Term() for _ in range(self.size)]

        idx = 0
        for i in range(self.cols):
            for e in self.sparse:
                if e.col != i:
                    continue

                sparse[idx].row = e.col
                sparse[idx].col = e.row
                sparse[idx].value = e.value
                idx += 1

        return MatrixSparse(
            rows=self.cols,
            cols=self.rows,
            size=self.size,
            sparse=spars,
        )
```

Arrays Sparse Matrix (8-1)

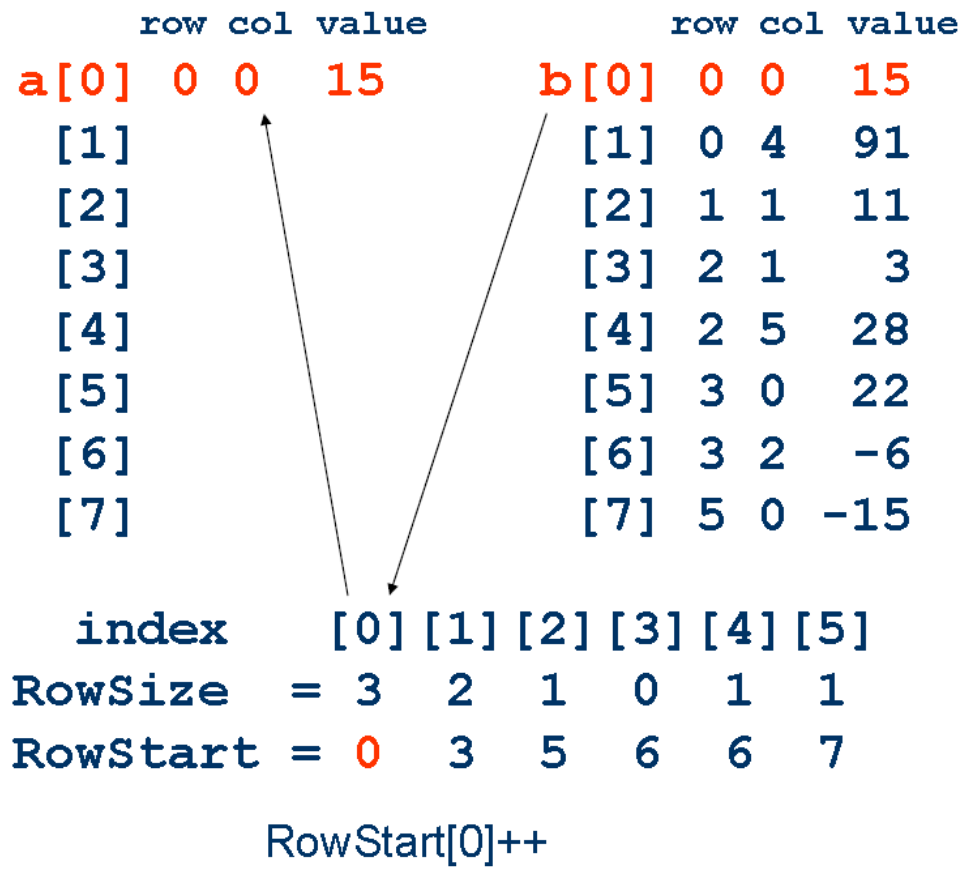
- Fast Transpose Algorithm
  - Determine the number of elements in each column of the original matrix.
  - Determine the starting positions of each row in the transpose matrix.

	[0]	[1]	[2]	[3]	[4]	[5]
RowSize =	3	2	1	0	1	1
RowStart =	0	3	5	6	6	7

	row	col	value
b[0]	0	0	15
[1]	0	4	91
[2]	1	1	11
[3]	2	1	3
[4]	2	5	28
[5]	3	0	22
[6]	3	2	-6
[7]	5	0	-15

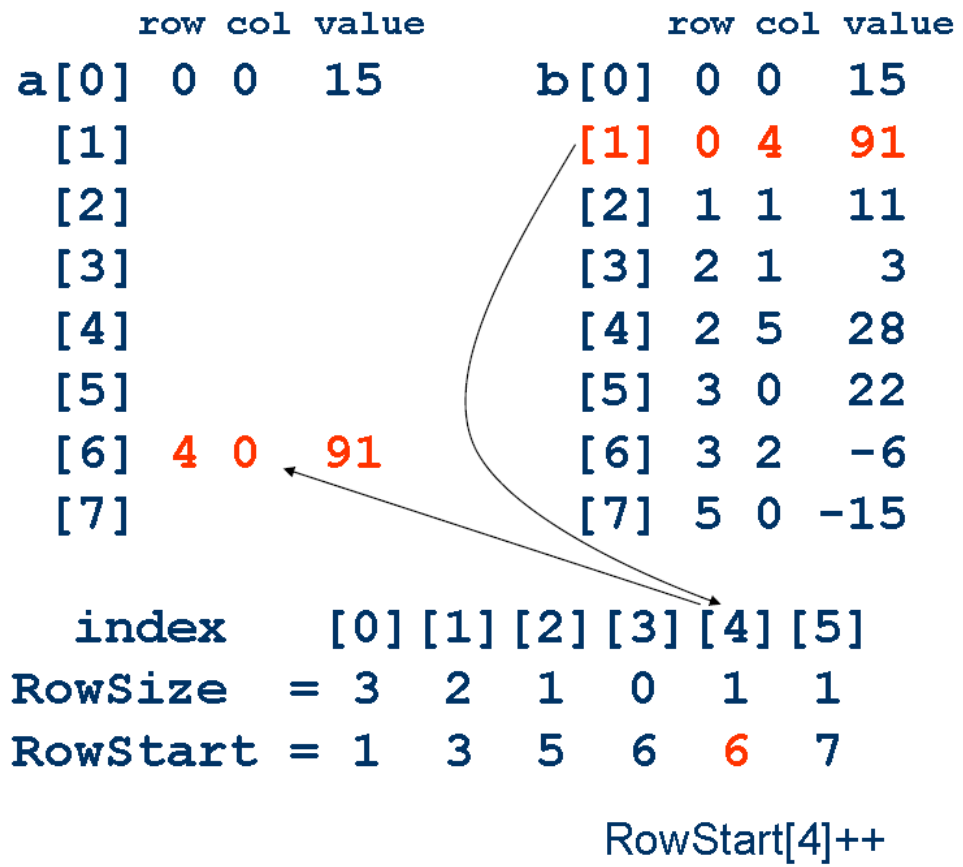
Arrays Sparse Matrix (8-2)

- Fast Transpose Algorithm



Arrays Sparse Matrix (8-3)

- Fast Transpose Algorithm



Arrays Sparse Matrix (8-4-Homework)

- Lab: matrix_spase.py
  - Sparse Matrix (4-lab)
  - “transpose_fast”

```
def transpose_fast(self):
   ….
```