人工智能学院

Data Structures and Algorithms

Multi-Dimensional Array2

Preview

1. Review
2. Practice

Review

◆ 数组的定义及特点：

定义：数组是由类型相同的数据元素构成的有序集合，每个元素称为数组元素，每个元素受n(n>=1)个线性关系的约束，可以通过下标访问该数据元素。

特点：数组可以看成是线性表的推广，其特点是结构中的元素本身可以是具有某种结构的数据， 但属于同一数据类型。简单来说：数组结构固定，数组行列数不可变、数据元素同构。

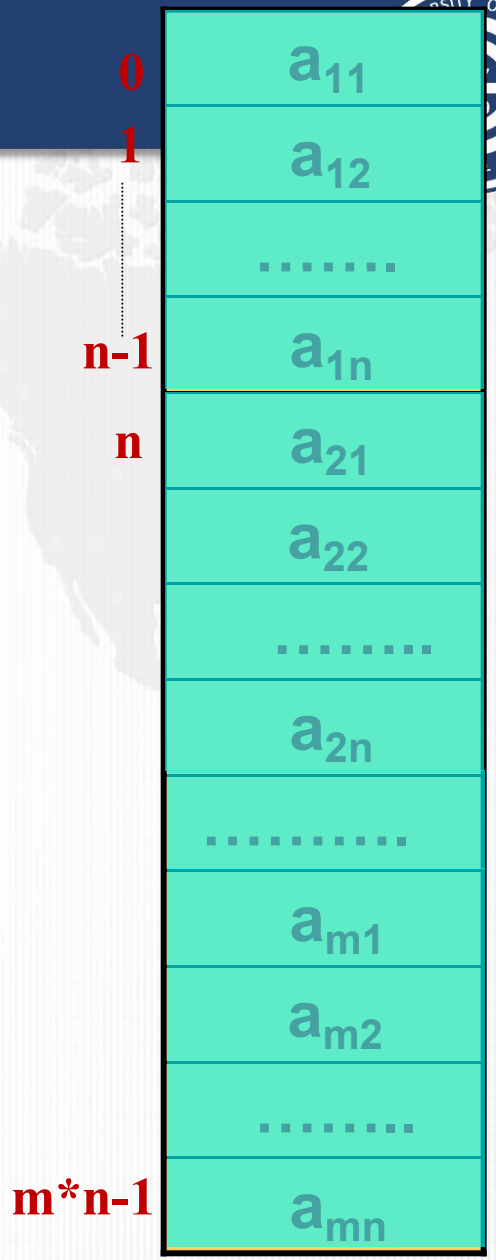
$$A_{m\times n}=\begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & \cdots & a_{mn} \end{bmatrix}$$

Review

◆ 次序约定：

- 以行序为主序：BASIC、PASCAL、C

a_{11}	a_{12}	.....	a_{1n}
a_{21}	a_{22}	.....	a_{2n}
.....			
a_{m1}	a_{m2}	.....	a_{mn}

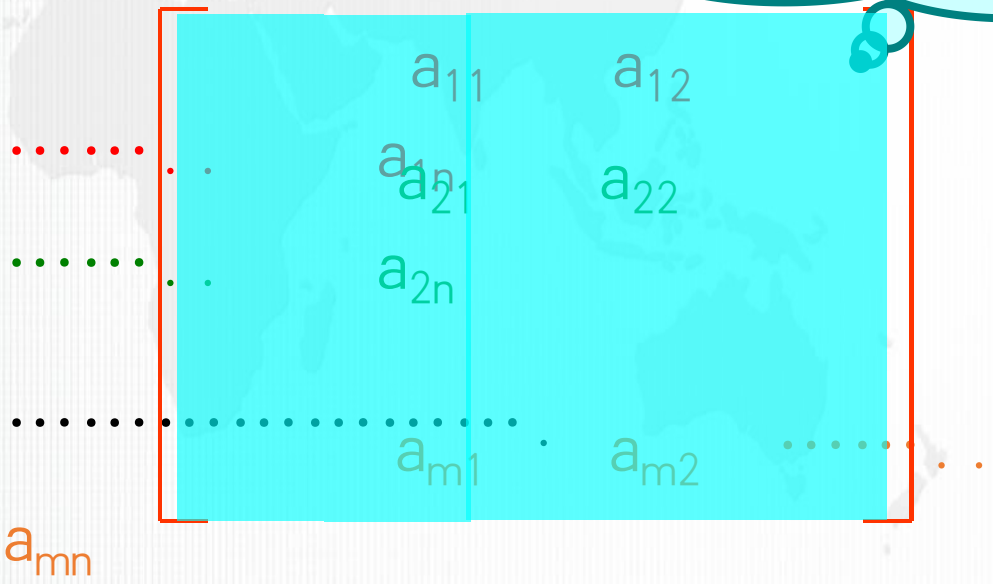


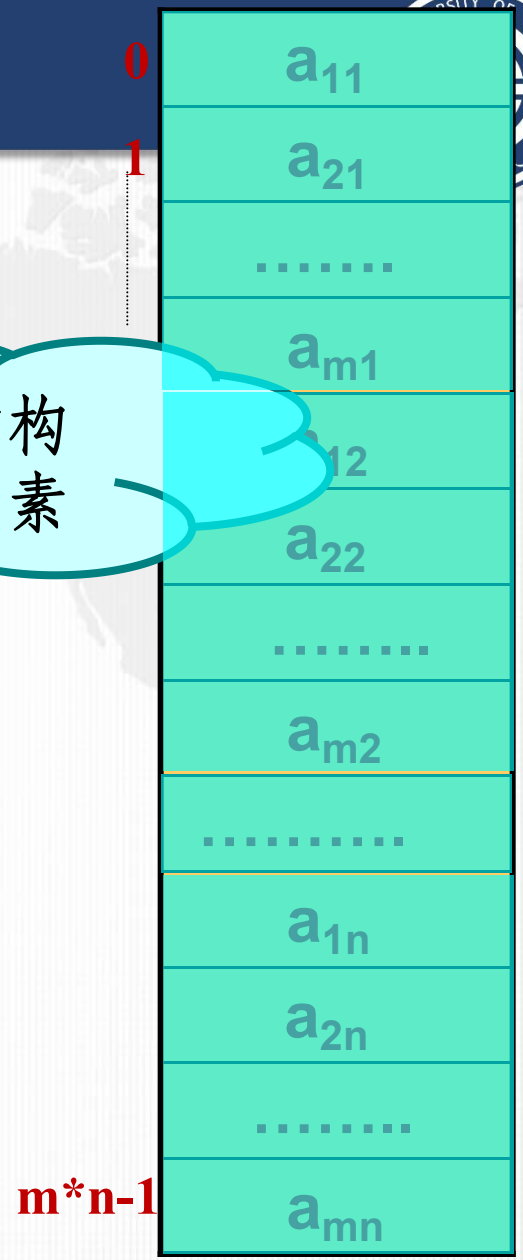
$$Loc(a_{ij}) = Loc(a_{11}) + [(i-1)n + (j-1)] * L$$

Review

- 次序约定：
  - 以行序为主序：BASIC、PASCAL、C
  - 以列序为主序：FORTRAN

数组的顺序存储结构
可随机存取任一元素

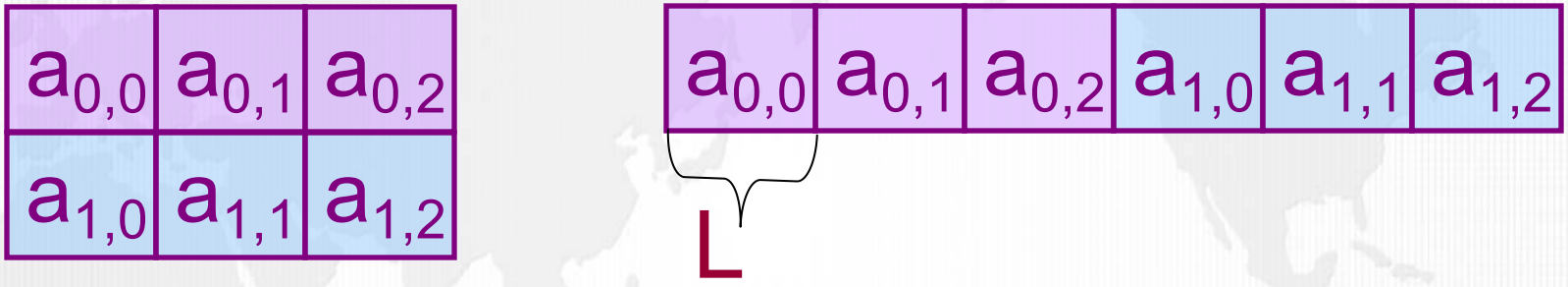




$$Loc(a_{ij})=Loc(a_{11})+[(j-1)m+(i-1)]*L$$

Review

◆ 以“行序为主序”的存储映象：



二维数组A中任一元素$a_{i,j}$的存储位置

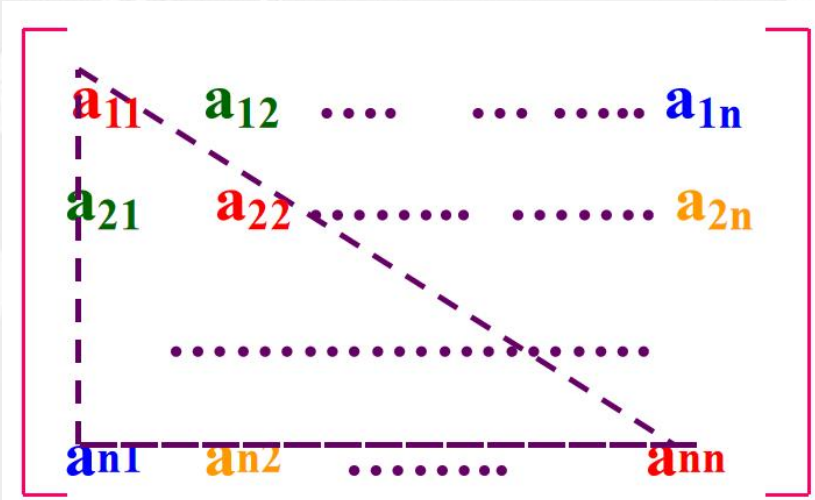
$$LOC(i,j) = LOC(0,0) + (b_2 \times i + j) \times L$$

推广到一般情况，可得到 n 维数组数据元素存储位置的映象关系

$$LOC(j_1, j_2, ..., j_n) = LOC(0,0,...,0) + (b_2 \times ... \times b_n \times j_1 + b_3 \times ... \times b_n \times j_2 + ...)$$

Review

◆ 对称矩阵：



按行序为主序

a_{11}
a_{21}
a_{22}
a_{31}
a_{32}
a_{33}
……
a_{n1}
……
a_{nn}

$a_{ij} = a_{ji}$

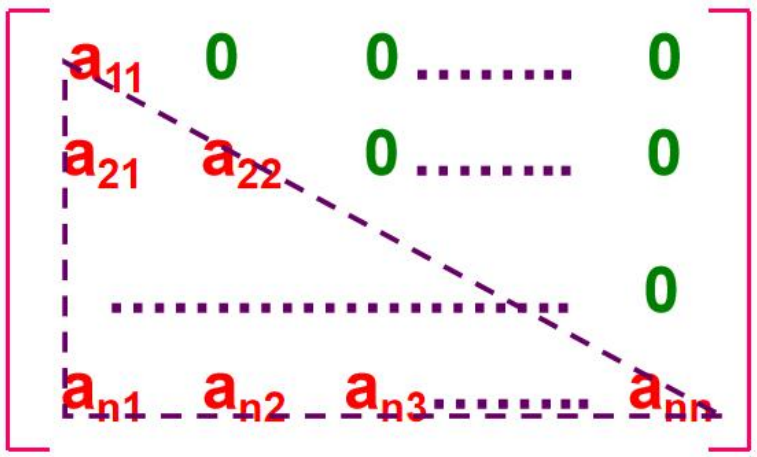
压缩存储元素个数：$1+2+\ldots+n=\frac{n(n+1)}{2}$

$Loc(a_{ij}) = Loc(a_{11}) + [\frac{i(i-1)}{2} + (j-1)]*L \quad (i \geq j)$

$Loc(a_{ij}) = Loc(a_{11}) + [\frac{j(j-1)}{2} + (i-1)]*L \quad (i < j)$

Review

◆ 三角矩阵：



按行序为主序

a_{11}
a_{21}
a_{22}
a_{31}
a_{32}
a_{33}
.....
a_{n1}
.....
a_{nn}

压缩存储元素个数：$1+2+\ldots+n=\frac{n(n+1)}{2}$

$Loc(a_{ij})= Loc(a_{11})+[\frac{i(i-1)}{2}+(j-1)]*L \quad (i\geq j)$

$a_{ij}= 0 \quad (i<j)$

Review

◆ 对角矩阵：

$$\begin{bmatrix} a_{11} & a_{12} & 0 & \cdots\cdots & 0 \\ a_{21} & a_{22} & a_{23} & 0 \cdots\cdots & 0 \\ 0 & a_{32} & a_{33} & a_{34} \cdots & 0 \\ & \cdots\cdots & & & \\ 0 & 0 \cdots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & \cdots & \cdots & a_{n,n-1} \quad a_{nn} \end{bmatrix}$$

按行序为主序

a_{11}
a_{12}
a_{21}
a_{22}
a_{23}
a_{32}
a_{33}
a_{34}
……
$a_{n\,n-1}$
a_{nn}

压缩存储元素个数：$(n-2)\times 3+4=3n-2$

$Loc(a_{ij})=Loc(a_{11})+[\,2(i-1)+(j-1)\,]*L$ （$|i-j|\leqslant 1$）

$a_{ij}=0$ 其它

Review

- **稀疏矩阵：**

  非零元较零元少，且分布没有一定规律的矩阵

  假设 m 行 n 列的矩阵含 t 个非零元素

  通常认为 δ ≤0.05 的矩阵为稀疏矩阵

$$\delta = \frac{t}{m \times n}$$

稀疏因子

- **矩阵的压缩存储基本思想：**

  1) 值相同的元素分配一个存储空间；

  2) 零元素不分配空间

M由{(1,2,12), (1,3,9), (3,1,-3),
(3,6,14), (4,3,24),(5,2,18),
(6,1,15), (6,4,-7) }
和矩阵维数（6,7）唯一确定

$$M = \begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}_{6\times7}$$

Review

◆ 三元组表：

行列下标　　非零元值

	i	j	e
0	6	7	8
1	1	2	12
2	1	3	9
3	3	1	-3
4	3	6	14
5	4	3	24
6	5	2	18
7	6	1	15
8	6	4	-7

ma.data

TSMatrix ma;

$$M=\begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}_{6\times 7}$$

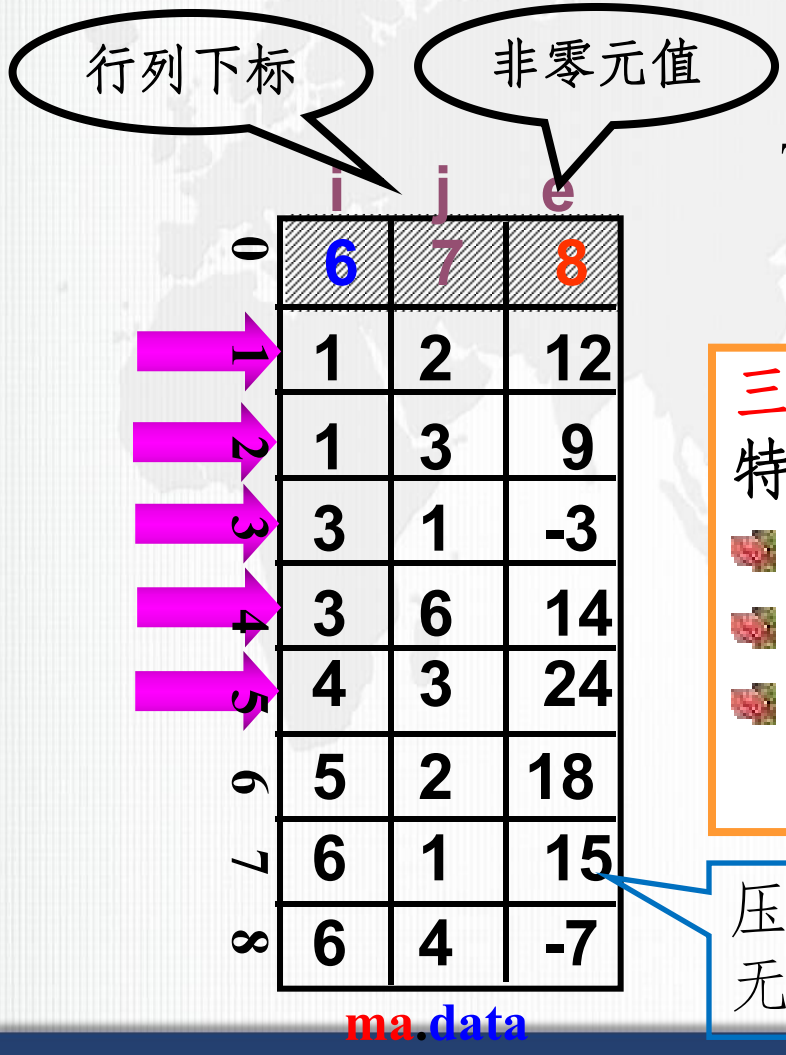
data[0]号单元未用
或存放矩阵行列维数和非零元个数

矩阵行数：　**ma.cols=6**
矩阵列数：　**ma.rows=7**
非零元个数：　**ma.elements=8**

三元组表所需存储单元个数为3(ma.elements+1)

Review

◆ 三元组表：



TSMatrix ma;

$$M=\begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}_{6\times 7}$$

三元组顺序表又称有序的双下标法

特点：

- 非零元在表中按行序有序存储
- 便于进行依行顺序处理的矩阵运算
- 若需存取某一行中的非零元，需从头开始查找。

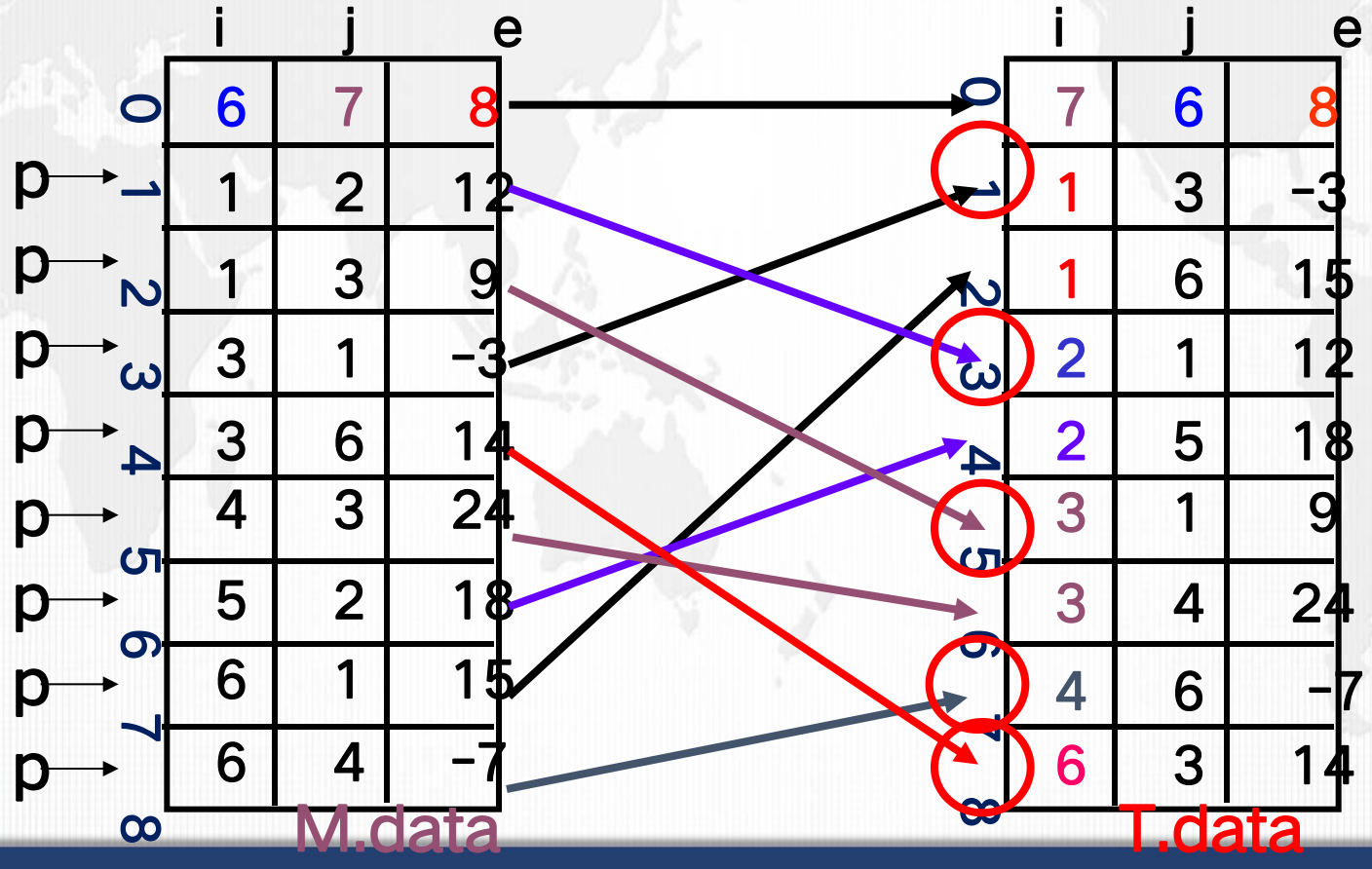
压缩存储后，元素a_{ij}的存储位置与其下标无关，而取决于之前的非零元个数

Review

- 快速转置算法：

1.按M.data中三元组次序转置，结果放入T.data中恰当位置

2.需要预先确定M中每一列第一个非零元在T.data中位置，



Review

◆ 快速转置算法：

1.按M.data中三元组次序转置，结果放入T.data中恰当位置
2.需要预先确定M中每一列第一个非零元在T.data中位置，为确定这些位置，应先求得M的每一列中非零元个数

实现：设两个数组

num[col]：矩阵M中第col列中非零元个数

cpot[col]：矩阵M中第col列第一个非零元在T.data中位置

显然有：

$$\begin{cases} cpot[1]=1; \\ cpot[col]= cpot[col-1]+ num[col-1]; \quad (2\le col\le M.nu) \end{cases}$$

Review

- 快速转置算法：

实现：设两个数组

num[col]：矩阵M中第col列中非零元个数

cpot[col]：矩阵M中第col列第一个非零元在T.data中位置

显然有：

$$\begin{cases} cpot[1]=1; \\ cpot[col]= cpot[col-1]+ num[col-1]; \quad (2\le col\le M.nu) \end{cases}$$

i j v

M.data

1	2	12
1	3	9
3	1	-3
3	6	14
4	3	24
5	2	18
6	1	15
6	4	-7

col	1	2	3	4	5	6	7
num[col]	2	2	2	1	0	1	0
cpot[col]	1	3	5	7	8	8	9

Review

- 快速转置算法：

col	1	2	3	4	5	6	7
num[col]	2	2	2	1	0	1	0
cpot[col]	1	3	5	7	8	8	9

2 4 6 9

3 5 7

M.data

	i	j	e
0	6	7	8
p→1	1	2	12
p→2	1	3	9
p→3	3	1	-3
p→4	3	6	14
p→5	4	3	24
p→6	5	2	18
p→7	6	1	15
p→8	6	4	-7

T.data

	i	j	e
0	7	6	8
1	1	3	-3
2	1	6	15
3	2	1	12
4	2	5	18
5	3	1	9
6	3	4	24
7	4	6	-7
8	6	3	14

Review

- **链式存储：**

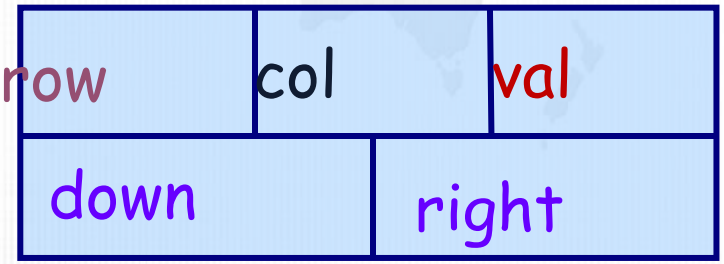
  引入链式存储的原因：

  ①用三元组表存储稀疏矩阵，在单纯的存储和做类似转置之类的运算时可以节约存储空间，且运算速度较快;

  ②但当进行矩阵相加等运算时，稀疏矩阵的非零元位置和个数都会发生变化。使用三元组表必然会引起数组元素　的大量移动。

- **十字链表：**

  每个非零元用含有五个域的结点表示（非零元的所在行、列、值，及同行、同列的下一个非零元）

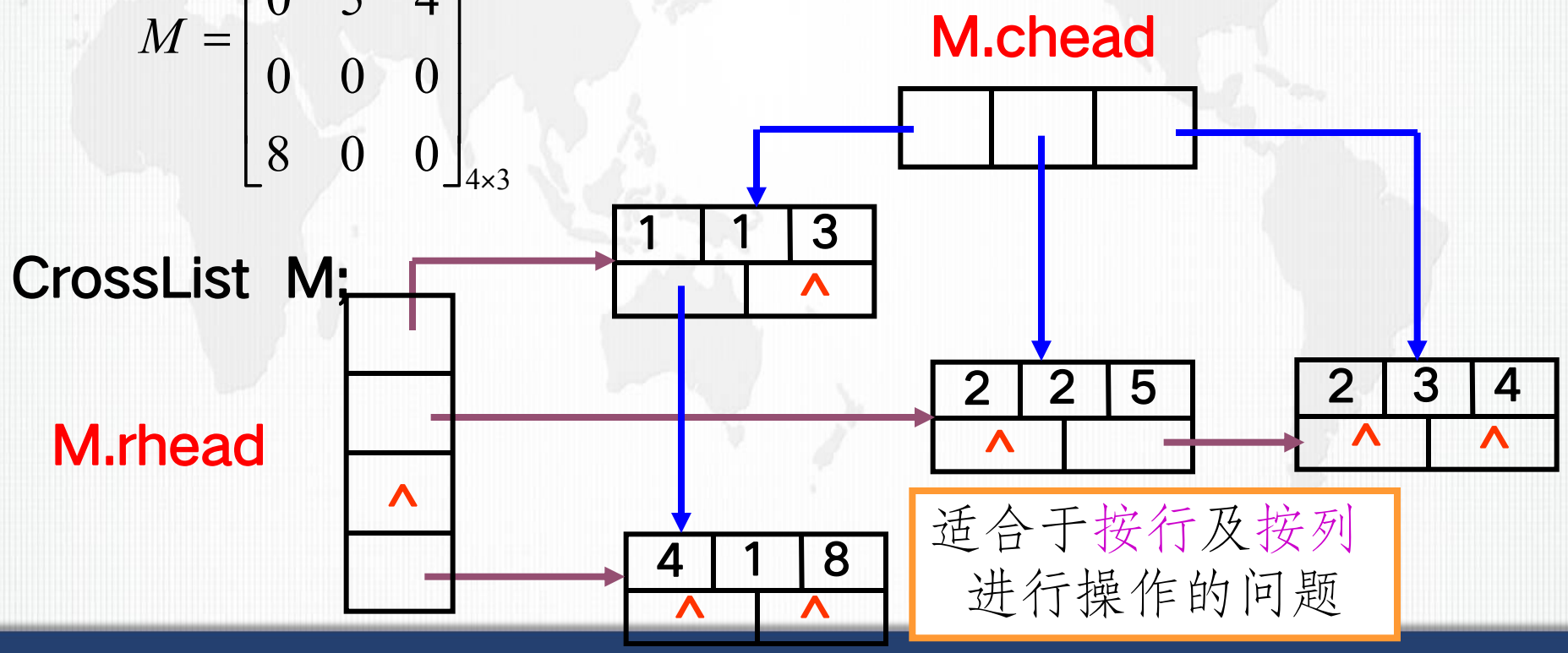


Review

- 十字链表：

  每个非零元用含有五个域的结点表示（非零元的所在行、列、值，及同行、同列的下一个非零元）

$$M=\begin{bmatrix}3&0&0\\0&5&4\\0&0&0\\8&0&0\end{bmatrix}_{4\times3}$$



Practice

◆ 练习1

给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出和为目标值 target 的那 两个 整数，并返回它们的数组下标。

你可以假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现

示例 1:

输入：nums = [2,7,11,15], target = 9

输出：[0,1]

解释：因为 nums[0] + nums[1] == 9 ，返回 [0, 1] 。

示例 2:

输入：nums = [3,2,4], target = 6

输出：[1,2]

示例 3:

输入：nums = [3,3], target = 6

输出：[0,1]

Practice

◆ 练习1

给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出 和为目标值 target 的那两个 整数，并返回它们的数组下标。

你可以假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现

方法一：

最容易想到的方法是枚举数组中的每一个数 x，寻找数组中是否存在 target - x。

当我们使用遍历整个数组的方式寻找 target - x 时，需要注意到每一个位于 x 之前的元素都已经和 x 匹配过，因此不需要再进行匹配。而每一个元素不能被使用两次，所以我们只需要在 x 后面的元素中寻找 target - x。

方法二：

采用哈希表的思路（后续查找会讲到）

Practice

◆ 练习2

给定一个长度为 n 的整数数组 height 。有 n 条垂线，第 i 条线的两个端点是 (i, 0) 和 (i, height[i]) 。找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。返回容器可以储存的最大水量。

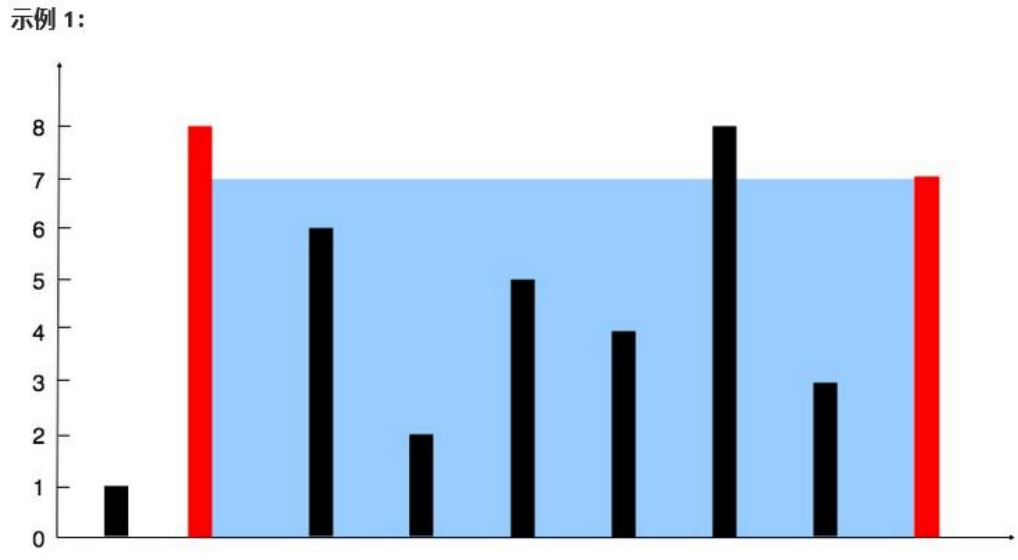
说明：你不能倾斜容器。

示例 1：

输入：[1,8,6,2,5,4,8,3,7]

输出：49

解释：图中垂直线代表输入数组 [1,8,6,2,5,4,8,3,7]。在此情况下，容器能够容纳水（表示为蓝色部分）的最大值为 49。



示例 2：

输入：height = [1,1]

输出：1

Practice

◆ 练习2

给定一个长度为 n 的整数数组 height 。有 n 条垂线，第 i 条线的两个端点是 (i, 0) 和 (i, height[i]) 。找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。返回容器可以储存的最大水量。

说明：你不能倾斜容器。

方法一：

暴力遍历

方法二：

在每个状态下，无论长板或短板向中间收窄一格，都会导致水槽底边宽度 −1 变短：

若向内移动短板，槽的短板min(h[i], h[j])可能变大，因此下个水槽的面积可能增大 。

若向内移动长板 ，水槽的短板min(h[i], h[j])不变或变小，下个水槽的面积一定变小 。

因此，初始化双指针分列水槽左右两端，循环每轮将短板向内移动一格，并更新面积最大值，直到两指针相遇时跳出；即可获得最大面积。

Practice

◆ 练习3

给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。

请必须使用时间复杂度为 O(log n) 的算法。

示例 1:

输入: nums = [1,3,5,6], target = 5

输出: 2

示例 2:

输入: nums = [1,3,5,6], target = 2

输出: 1

示例 3:

输入: nums = [1,3,5,6], target = 7

输出: 4

Practice

◆ 练习3

给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。

请必须使用时间复杂度为 O(logn) 的算法。

如果该题目暴力解决的话需要 O(n)的时间复杂度，但是如果二分的话则可以降低到O(logn)的时间复杂度：

整体思路和普通的二分查找几乎没有区别，

1. 先设定左侧下标 left 和右侧下标 right，再计算中间下标 mid
2. 每次根据 nums[mid] 和 target 之间的大小进行判断，相等则直接返回下标，nums[mid] < target 则 left 右移，nums[mid] > target 则 right 左移
3. 查找结束如果没有相等值则返回 left，该值为插入位置

Practice

◆ 练习4

给你一个 升序排列 的数组 nums ，请你 原地 删除重复出现的元素，使每个元素 只出现一次 ，返回删除后数组的新长度。元素的 相对顺序 应该保持 一致 。

不要使用额外的空间，你必须在 原地 修改输入数组 并在使用 O(1) 额外空间的条件下完成。

示例 1：

输入：nums = [1,1,2]

输出：2, nums = [1,2]

解释：函数应该返回新的长度 2 ，并且原数组 nums 的前两个元素被修改为 1, 2 。不需要考虑数组中超出新长度后面的元素。

示例 2：

输入：nums = [0,0,1,1,1,2,2,3,3,4]

输出：5, nums = [0,1,2,3,4]

解释：函数应该返回新的长度 5 ， 并且原数组 nums 的前五个元素被修改为 0, 1, 2, 3, 4 。不需要考虑数组中超出新长度后面的元素。

Practice

◆ 练习4

给你一个 升序排列 的数组 nums，请你 原地 删除重复出现的元素，使每个元素 只出现一次，返回删除后数组的新长度。元素的 相对顺序 应该保持 一致 。

不要使用额外的空间，你必须在 原地 修改输入数组 并在使用 O(1) 额外空间的条件下完成。

首先注意数组是有序的，那么重复的元素一定会相邻。

要求删除重复元素，实际上就是将不重复的元素移到数组的左侧。

考虑用 2 个指针，一个在前记作 p，一个在后记作 q，算法流程如下：

1.比较 p 和 q 位置的元素是否相等。

- 如果相等，q 后移 1 位
- 如果不相等，将 q 位置的元素复制到 p+1 位置上，p 后移一位，q 后移 1 位
- 重复上述过程，直到 q 等于数组长度。

2.返回 p + 1，即为新数组长度。

数组的应用举例

◆ 练习5

给你一个数组 nums 和一个值 val，你需要 原地 移除所有数值等于 val 的元素，并返回移除后数组的新长度。

不要使用额外的数组空间，你必须仅使用 O(1) 额外空间并 原地 修改输入数组。

示例 1：

输入：nums = [3,2,2,3], val = 3

输出：2, nums = [2,2]

解释：函数应该返回新的长度 2, 并且 nums 中的前两个元素均为 2。你不需要考虑数组中超出新长度后面的元素。例如，函数返回的新长度为 2 ，而 nums = [2,2,3,3] 或 nums = [2,2,0,0]，也会被视作正确答案。

示例 2：

输入：nums = [0,1,2,2,3,0,4,2], val = 2

输出：5, nums = [0,1,4,0,3]

解释：函数应该返回新的长度 5, 并且 nums 中的前五个元素为 0, 1, 3, 0, 4。注意这五个元素可为任意顺序。你不需要考虑数组中超出新长度后面的元素。

数组的应用举例

◆ 练习5

给你一个数组 nums 和一个值 val，你需要 原地 移除所有数值等于 val 的元素，并返回移除后数组的新长度。

不要使用额外的数组空间，你必须仅使用 O(1) 额外空间并 原地 修改输入数组。

由于题目要求删除数组中等于 val的元素，因此输出数组的长度一定小于等于输入数组的长度，我们可以把输出的数组直接写在输入数组上。

可以使用双指针：右指针 right 指向当前将要处理的元素，左指针 left指向下一个将要赋值的位置。

- 如果右指针指向的元素不等于 val它一定是输出数组的一个元素，我们就将右指针指向的元素复制到左指针位置，然后将左右指针同时右移；
- 如果右指针指向的元素等于 val，它不能在输出数组里，此时左指针不动，右指针右移一位。
- 整个过程保持不变的性质是：区间[0, left) 中的元素都不等于 val。当左右指针遍历完输入数组以后，left 的值就是输出数组的长度。

这样的算法在最坏情况下（输入数组中没有元素等于 val），左右指针各遍历了数组一次

Homework

- LeetCode题目全部AC

Thanks!

See you in the next session!