山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：202200400053 | 姓名： 王宇涵 | | 班级： 22级2班 |
| 实验题目：数组与矩阵 | | | |
| 实验学时：2 | | 实验日期： 2023-10-18 | |
| 实验目的：  掌握稀疏矩阵结构的描述及操作的实现。 | | | |
| 软件开发环境：  Vscode | | | |
| 1. 实验内容   创建 **稀疏矩阵类** （参照课本 MatrixTerm 三元组定义） ,采用行主顺序把稀疏矩阵非 0 元素映射到一维数组中,提供操作:两个稀疏矩阵相加、两个稀疏矩阵相乘、稀疏矩阵的转置、输出矩阵。  键盘输入矩阵的行数、列数;并按行优先顺序输入矩阵的各元素值,建立矩阵;  对建立的矩阵执行相加、相乘、转置的操作,输出操作的结果矩阵。   1. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）   **整体思路**  当进入程序的 main 函数后，程序首先读取一个整数 w，表示操作的次数。接下来，进入一个循环，循环次数为 w，每次迭代都根据用户的选择执行不同的操作。  根据用户的选择 op，程序执行以下不同的操作：  op == 1：  用户选择初始化稀疏矩阵。程序会读取矩阵的行数 n 和列数 m，然后初始化一个稀疏矩阵 s，接着读取矩阵的元素值并将其存储在 s 中。  op == 2：  用户选择进行矩阵乘法操作。程序会读取两个矩阵的行数 n 和列数 m，以及非零元素的数量 t。首先，初始化一个右矩阵 tmp，然后读取右矩阵的非零元素并存储在 tmp 中。接着，执行稀疏矩阵 s 与右矩阵 tmp 的矩阵乘法操作，并将结果存储在 s 中。  op == 3：  用户选择进行矩阵加法操作。程序会读取两个矩阵的行数 n 和列数 m，以及非零元素的数量 t。首先，初始化一个右矩阵 tmp，然后读取右矩阵的非零元素并存储在 tmp 中。接着，执行稀疏矩阵 s 与右矩阵 tmp 的矩阵加法操作，并将结果存储在 s 中。  op == 4：  用户选择输出当前稀疏矩阵 s。  op == 5：  用户选择进行转置操作，将当前稀疏矩阵 s 进行转置。  每次操作完成后，根据用户的选择，程序会输出相应的结果或错误信息。这个程序允许用户对稀疏矩阵进行不同的操作，包括初始化、矩阵乘法、矩阵加法、输出和转置等。操作结果会根据用户的选择输出到标准输出  **难点在于矩阵乘法,转置和加法**  **矩阵乘法的主要思路**  首先，检查左矩阵的列数是否等于右矩阵的行数。如果不相等，无法进行矩阵乘法，输出错误信息（这部分的代码在 if (cols != b.rows) 处）。  创建一个新的稀疏矩阵 c 用于存储结果。  初始化辅助数组，例如 rowSize 用于存储右矩阵每一行的非零元素数量，以及 nextRow 用于记录每一行的非零元素在右矩阵 b 中的起始位置。  遍历左矩阵 terms 中的元素，对于每个元素 (row, col, value)，找到右矩阵 b 中行号等于 col 的元素，并将它们的值相乘累加，得到结果矩阵 c 中的元素。  将结果矩阵 c 赋值给当前矩阵对象，即 \*this = c。  这样，经过上述步骤，左矩阵 s 与右矩阵 b 的乘法结果将存储在当前矩阵 s 中。  **矩阵转置的主要思路**  首先，创建一个新的稀疏矩阵 b 用于存储转置后的矩阵。  初始化一些辅助数组，例如 colSize 用于记录每一列的非零元素数量，以及 nextRow 用于记录每一列中非零元素在转置后矩阵中的位置。  遍历原矩阵 s 中的每一个元素 (row, col, value)，将其加入到转置矩阵 b 中，并根据列号 col 在 nextRow 中找到合适的位置进行插入。  最后，将转置矩阵 b 赋值给当前矩阵对象，即 \*this = b。  通过这样的操作，原矩阵 s 就被成功地转置成了新的矩阵 b。  **矩阵加法的主要思路**  检查两个矩阵的维度是否兼容，即它们的行数和列数是否相同。如果它们的行数和列数不相同，则无法进行矩阵加法，程序输出错误信息。  创建一个新的稀疏矩阵 c 用于存储矩阵相加的结果。  初始化两个指向两个输入矩阵 s 和 b 中非零元素的指针 i 和 j。  使用循环遍历两个输入矩阵中的非零元素，按照行主的顺序比较它们的行和列：  a. 如果 s 的当前元素小于 b 的当前元素，将 s 的元素添加到结果矩阵 c 中，并将 s 的指针 i 向前移动一步。  b. 如果 b 的当前元素小于 s 的当前元素，将 b 的元素添加到结果矩阵 c 中，并将 b 的指针 j 向前移动一步。  c. 如果两个当前元素相等，将它们相加，并将结果添加到结果矩阵 c 中。然后，同时将 s 和 b 的指针 i 和 j 向前移动一步。  继续循环，直到遍历完两个输入矩阵的所有非零元素。  最后，将结果矩阵 c 赋值给当前矩阵对象，即 \*this = c。   1. 测试结果（测试输入，测试输出）   输入1  7  1  5 5  2 1 0 0 0  0 0 -1 0 0  0 0 0 0 0  0 0 -1 0 0  0 0 0 0 0  3  5 5  4  2 2 5  3 5 8  4 4 2  5 3 4  4  2  5 5  3  1 1 8  2 4 4  3 5 2  4  5  4  输出1  5 5  2 1 0 0 0  0 5 -1 0 0  0 0 0 0 8  0 0 -1 2 0  0 0 4 0 0  5 5  16 0 0 4 0  0 0 0 20 -2  0 0 0 0 0  0 0 0 0 -2  0 0 0 0 8  5 5  16 0 0 0 0  0 0 0 0 0  0 0 0 0 0  4 20 0 0 0  0 -2 0 -2 8  输入2  40  1  10 20  -1 0 1 0 0 0 0 0 -1 0 0 0 -1 0 -1 0 0 -1 1 -1  0 0 2 -1 0 0 0 0 0 -1 0 0 0 0 0 0 0 1 -2 0  1 0 0 0 0 0 0 0 0 0 -1 -2 -1 0 -1 0 0 0 0 0  0 0 0 1 0 -1 -1 -1 0 0 1 0 0 0 0 0 0 0 -1 0  0 0 0 1 0 0 0 0 0 1 -1 1 0 0 0 0 -1 0 0 0  1 0 -1 1 2 0 0 0 1 0 0 0 0 -1 0 1 -1 1 -1 0  -1 0 0 0 -1 0 0 0 0 0 -1 0 0 -1 2 0 0 -1 0 0  -1 -1 -1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0  0 0 0 0 0 0 -3 0 0 0 0 -1 -2 1 0 2 0 -1 -1 0  -1 1 0 1 -1 0 0 0 -1 0 -1 0 0 0 0 1 0 0 -1 1  2  10 20  7  2 16 9  3 7 3  3 17 4  6 3 4  7 12 10  8 13 6  10 8 3  2  10 20  8  1 20 1  4 20 5  6 5 4  6 10 10  7 4 8  7 6 10  8 12 9  9 17 5  2  10 20  9  1 8 4  3 8 6  3 17 7  5 1 10  5 8 4  6 9 4  7 12 7  9 10 9  9 17 7  3  10 20  7  3 3 10  5 18 4  8 5 2  8 19 5  8 20 10  9 12 3  10 11 10  4  2  10 20  2  3 16 4  4 10 6  2  10 20  7  1 16 8  2 9 8  3 8 9  4 2 4  4 20 7  8 10 7  10 3 4  2  10 20  1  1 19 5  2  10 20  10  1 9 8  2 15 5  3 2 10  4 2 5  4 3 9  4 7 10  6 6 6  6 14 6  7 2 7  9 16 9  2  10 20  7  3 14 5  4 9 8  6 19 5  7 17 7  8 13 4  9 6 10  9 20 1  5  2  20 10  7  6 9 2  7 8 10  7 9 9  11 1 10  12 5 6  18 4 8  20 6 4  2  20 10  2  13 2 5  17 5 10  1  19 19  2 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 1 0 1  0 0 1 0 1 -3 0 0 -1 1 -2 0 -2 0 0 1 0 0 0  -1 -1 0 0 1 0 0 1 0 -1 0 0 1 1 0 0 0 1 0  0 0 -1 0 0 -2 -1 0 0 0 1 0 0 1 2 -1 2 0 0  0 1 0 -1 0 0 -1 0 0 -1 0 0 0 -1 0 -1 0 -1 0  0 0 0 -1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 -1  1 0 -1 1 0 0 -1 0 1 1 0 0 0 0 1 0 1 0 -1  0 0 0 1 0 0 0 -1 0 0 0 0 0 0 0 0 -1 0 -1  0 -1 1 0 0 0 0 0 0 0 -1 0 0 0 -1 1 0 0 0  0 0 -1 0 0 0 0 1 0 -1 2 0 2 -1 -1 0 -1 0 0  1 0 0 -1 0 0 0 0 0 0 0 0 0 0 -1 1 0 0 0  0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 -1  -1 0 0 0 2 -1 2 -2 0 0 0 -1 1 0 0 0 0 0 0  0 0 1 0 0 -1 0 0 0 0 0 0 0 0 0 -1 0 0 -2  0 0 -1 0 0 1 0 0 1 -1 0 0 0 1 0 0 0 0 1  0 0 0 0 1 -1 -1 1 1 0 0 0 0 -1 0 0 0 0 0  0 -1 0 0 0 0 2 0 0 0 0 2 2 0 0 0 0 0 -1  0 0 0 -1 0 -1 0 0 0 0 0 -1 0 0 0 0 0 0 0  0 -1 -1 0 0 1 0 -1 1 0 -1 0 0 0 0 0 0 0 1  4  2  19 19  6  5 5 2  5 17 5  12 3 3  13 15 5  14 3 5  15 9 7  2  19 19  8  7 9 1  10 1 6  12 2 4  14 3 9  14 8 2  16 7 3  18 1 1  18 14 4  2  19 19  9  1 5 3  1 18 10  4 15 4  6 7 9  11 19 6  12 2 1  14 7 6  14 14 2  17 9 8  2  19 19  7  4 18 7  5 9 1  7 2 6  11 9 3  12 16 3  15 9 2  16 5 5  2  19 19  3  3 12 4  17 7 5  18 16 4  5  2  19 19  1  17 17 2  3  19 19  6  11 8 5  11 14 5  12 19 6  17 5 4  17 15 6  19 19 4  2  19 19  7  1 1 4  4 12 5  6 1 9  7 8 3  9 18 8  13 12 2  16 14 2  2  19 19  2  8 11 7  12 4 8  3  19 19  7  1 16 5  3 9 6  5 15 3  14 14 10  15 9 6  15 14 3  15 19 7  2  19 19  6  1 19 2  5 8 6  6 16 6  9 6 6  10 18 9  15 7 5  5  5  2  19 19  6  6 7 1  10 7 6  13 5 5  15 16 6  17 9 10  19 15 3  2  19 19  6  3 5 4  4 9 5  5 15 1  11 3 5  17 5 6  17 7 7  2  19 19  1  14 6 7  5  2  19 19  3  3 10 8  4 18 1  15 15 8  2  19 19  4  5 8 10  6 9 10  6 16 6  14 15 4  5  4  2  19 19  9  2 17 3  4 18 9  12 3 8  13 11 10  13 19 7  14 12 4  15 4 9  17 8 9  19 4 5  2  19 19  1  7 17 6  输出2  -1  -1  -1  10 20  0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 10 0 0 0 0 6 0 0 0 0 0 0 0 0 7 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  10 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 4 0 0  0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0  0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 5 10  0 0 0 0 0 0 0 0 0 9 0 3 0 0 0 0 7 0 0 0  0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0  -1  -1  -1  -1  -1  -1  -1  19 19  2 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 1 0 1  0 0 1 0 1 -3 0 0 -1 1 -2 0 -2 0 0 1 0 0 0  -1 -1 0 0 1 0 0 1 0 -1 0 0 1 1 0 0 0 1 0  0 0 -1 0 0 -2 -1 0 0 0 1 0 0 1 2 -1 2 0 0  0 1 0 -1 0 0 -1 0 0 -1 0 0 0 -1 0 -1 0 -1 0  0 0 0 -1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 -1  1 0 -1 1 0 0 -1 0 1 1 0 0 0 0 1 0 1 0 -1  0 0 0 1 0 0 0 -1 0 0 0 0 0 0 0 0 -1 0 -1  0 -1 1 0 0 0 0 0 0 0 -1 0 0 0 -1 1 0 0 0  0 0 -1 0 0 0 0 1 0 -1 2 0 2 -1 -1 0 -1 0 0  1 0 0 -1 0 0 0 0 0 0 0 0 0 0 -1 1 0 0 0  0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 -1  -1 0 0 0 2 -1 2 -2 0 0 0 -1 1 0 0 0 0 0 0  0 0 1 0 0 -1 0 0 0 0 0 0 0 0 0 -1 0 0 -2  0 0 -1 0 0 1 0 0 1 -1 0 0 0 1 0 0 0 0 1  0 0 0 0 1 -1 -1 1 1 0 0 0 0 -1 0 0 0 0 0  0 -1 0 0 0 0 2 0 0 0 0 2 2 0 0 0 0 0 -1  0 0 0 -1 0 -1 0 0 0 0 0 -1 0 0 0 0 0 0 0  0 -1 -1 0 0 1 0 -1 1 0 -1 0 0 0 0 0 0 0 1  19 19  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  输入3  20  1  5 11  -22324 -8307 9206 122 -7218 21649 -16209 11639 3813 12960 15895  -6355 8061 -4443 9028 -2663 20150 6485 8100 -12939 -1189 -8954  17884 -3031 -10317 6894 9240 -1078 9344 -16194 -1543 -16063 -15494  -19732 3868 -25565 1922 4300 8148 -13256 4611 2077 26163 10738  10610 -2944 6357 4205 -12046 2795 13566 18396 11768 -5985 -3455  2  5 5  25  1 1 1  1 2 2  1 3 2  1 4 2  1 5 3  2 1 1  2 2 1  2 3 3  2 4 2  2 5 2  3 1 1  3 2 1  3 3 3  3 4 2  3 5 1  4 1 2  4 2 1  4 3 3  4 4 2  4 5 2  5 1 2  5 2 2  5 3 3  5 4 2  5 5 2  4  2  5 5  25  1 1 2  1 2 1  1 3 2  1 4 1  1 5 2  2 1 3  2 2 2  2 3 1  2 4 3  2 5 1  3 1 1  3 2 1  3 3 1  3 4 2  3 5 3  4 1 2  4 2 1  4 3 2  4 4 3  4 5 3  5 1 2  5 2 2  5 3 2  5 4 1  5 5 3  2  5 5  25  1 1 3  1 2 1  1 3 1  1 4 2  1 5 1  2 1 1  2 2 1  2 3 2  2 4 2  2 5 3  3 1 3  3 2 1  3 3 3  3 4 1  3 5 1  4 1 2  4 2 3  4 3 2  4 4 3  4 5 1  5 1 1  5 2 1  5 3 2  5 4 1  5 5 1  2  5 5  25  1 1 2  1 2 2  1 3 1  1 4 3  1 5 2  2 1 1  2 2 3  2 3 3  2 4 1  2 5 1  3 1 2  3 2 2  3 3 2  3 4 1  3 5 3  4 1 3  4 2 1  4 3 1  4 4 2  4 5 2  5 1 2  5 2 2  5 3 1  5 4 1  5 5 3  2  5 5  25  1 1 2  1 2 1  1 3 1  1 4 2  1 5 2  2 1 2  2 2 3  2 3 1  2 4 2  2 5 2  3 1 3  3 2 1  3 3 3  3 4 3  3 5 2  4 1 3  4 2 2  4 3 3  4 4 2  4 5 1  5 1 1  5 2 2  5 3 3  5 4 2  5 5 2  4  2  5 5  25  1 1 2  1 2 3  1 3 2  1 4 2  1 5 3  2 1 1  2 2 3  2 3 2  2 4 1  2 5 2  3 1 1  3 2 1  3 3 2  3 4 3  3 5 1  4 1 1  4 2 2  4 3 1  4 4 1  4 5 1  5 1 3  5 2 2  5 3 2  5 4 1  5 5 2  2  5 5  25  1 1 3  1 2 2  1 3 2  1 4 3  1 5 2  2 1 3  2 2 2  2 3 2  2 4 2  2 5 3  3 1 3  3 2 3  3 3 1  3 4 3  3 5 1  4 1 2  4 2 3  4 3 1  4 4 3  4 5 1  5 1 3  5 2 3  5 3 3  5 4 3  5 5 2  3  5 5  25  1 1 2  1 2 2  1 3 1  1 4 3  1 5 2  2 1 1  2 2 1  2 3 3  2 4 3  2 5 1  3 1 2  3 2 1  3 3 3  3 4 2  3 5 1  4 1 2  4 2 3  4 3 2  4 4 1  4 5 2  5 1 1  5 2 1  5 3 3  5 4 1  5 5 2  4  4  2  12 13  156  1 1 3  1 2 2  1 3 2  1 4 3  1 5 3  1 6 3  1 7 1  1 8 2  1 9 3  1 10 2  1 11 1  1 12 3  1 13 3  2 1 1  2 2 1  2 3 2  2 4 3  2 5 2  2 6 2  2 7 3  2 8 1  2 9 1  2 10 2  2 11 1  2 12 1  2 13 2  3 1 2  3 2 2  3 3 2  3 4 2  3 5 1  3 6 3  3 7 3  3 8 2  3 9 2  3 10 3  3 11 2  3 12 3  3 13 2  4 1 3  4 2 2  4 3 2  4 4 2  4 5 3  4 6 2  4 7 2  4 8 3  4 9 2  4 10 2  4 11 2  4 12 2  4 13 3  5 1 2  5 2 1  5 3 3  5 4 3  5 5 3  5 6 2  5 7 3  5 8 2  5 9 1  5 10 3  5 11 2  5 12 3  5 13 3  6 1 1  6 2 3  6 3 3  6 4 2  6 5 1  6 6 3  6 7 2  6 8 3  6 9 2  6 10 1  6 11 3  6 12 2  6 13 3  7 1 2  7 2 2  7 3 3  7 4 1  7 5 1  7 6 1  7 7 2  7 8 1  7 9 3  7 10 1  7 11 1  7 12 3  7 13 3  8 1 1  8 2 2  8 3 1  8 4 3  8 5 3  8 6 2  8 7 3  8 8 1  8 9 2  8 10 1  8 11 2  8 12 3  8 13 1  9 1 3  9 2 3  9 3 2  9 4 1  9 5 3  9 6 3  9 7 3  9 8 1  9 9 1  9 10 3  9 11 2  9 12 2  9 13 2  10 1 3  10 2 3  10 3 1  10 4 1  10 5 1  10 6 3  10 7 2  10 8 1  10 9 1  10 10 3  10 11 3  10 12 3  10 13 2  11 1 1  11 2 2  11 3 2  11 4 3  11 5 2  11 6 1  11 7 1  11 8 2  11 9 3  11 10 2  11 11 3  11 12 2  11 13 3  12 1 2  12 2 3  12 3 1  12 4 2  12 5 2  12 6 2  12 7 3  12 8 3  12 9 2  12 10 2  12 11 1  12 12 1  12 13 2  2  5 5  25  1 1 1  1 2 3  1 3 3  1 4 1  1 5 3  2 1 2  2 2 2  2 3 3  2 4 2  2 5 1  3 1 3  3 2 1  3 3 1  3 4 2  3 5 1  4 1 1  4 2 3  4 3 1  4 4 1  4 5 3  5 1 3  5 2 2  5 3 1  5 4 2  5 5 1  2  5 5  25  1 1 2  1 2 1  1 3 2  1 4 1  1 5 2  2 1 1  2 2 1  2 3 2  2 4 2  2 5 2  3 1 1  3 2 1  3 3 1  3 4 3  3 5 3  4 1 2  4 2 1  4 3 3  4 4 3  4 5 3  5 1 2  5 2 1  5 3 2  5 4 1  5 5 1  4  2  5 5  25  1 1 2  1 2 3  1 3 2  1 4 3  1 5 3  2 1 1  2 2 2  2 3 3  2 4 3  2 5 2  3 1 3  3 2 1  3 3 3  3 4 2  3 5 1  4 1 3  4 2 1  4 3 3  4 4 2  4 5 3  5 1 1  5 2 1  5 3 3  5 4 3  5 5 2  4  2  5 5  25  1 1 3  1 2 1  1 3 2  1 4 2  1 5 1  2 1 1  2 2 3  2 3 3  2 4 3  2 5 1  3 1 1  3 2 2  3 3 1  3 4 1  3 5 3  4 1 2  4 2 2  4 3 1  4 4 3  4 5 1  5 1 2  5 2 2  5 3 3  5 4 2  5 5 1  输出3  -1  5 5  1 2 2 2 3  1 1 3 2 2  1 1 3 2 1  2 1 3 2 2  2 2 3 2 2  5 5  16143 13975 16499 16583 13958  14052 12162 14360 14438 12152  12440 10759 12704 12777 10751  15373 13308 15713 15794 13293  17168 14856 17540 17632 14838  5 5  1945386 1782533 1254468 1903751 1285027  1693395 1551629 1091979 1657151 1118577  1498468 1373016 966290 1466390 989824  1852660 1697571 1194674 1813009 1223776  2068355 1895207 1333773 2024085 1366260  5 5  1945386 1782533 1254468 1903751 1285027  1693395 1551629 1091979 1657151 1118577  1498468 1373016 966290 1466390 989824  1852660 1697571 1194674 1813009 1223776  2068355 1895207 1333773 2024085 1366260  -1  -1  5 5  16 11 20 22 23  15 10 19 22 24  14 8 17 15 18  14 9 18 16 17  15 9 19 17 20  5 5  192 135 260 234 202  187 130 255 229 198  150 108 202 184 156  156 111 208 188 160  167 119 225 204 173   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   问题:超时  原因和解决方法  1. 没有考虑到按行主列存储,所以每次进行输出,之前的乘法函数都是从头到尾枚举矩阵的全部三元组,改进则直接使用指针的方法往后索引即可.重新实现了乘法函数和输出函数.  2. 每次进行push\_back都需要重新申请空间,因此直接初始化terms的size为一个很大的数就可以用空间换时间,重定义了reSetSize函数.  3. 使用scanf相对比较耗时,最后使用scanf替代cin通过了最后一个测试点.   1. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）   #include<iostream>  using namespace std;  const int maxsize=1e5;  template <class T>  class LinearList  {  public:  virtual ~LinearList(){};  virtual bool empty()const =0;  virtual int size()const =0;  virtual T get(int theIndex)const =0;  virtual int indexOf(const T& x)const =0;  virtual void erase(int theIndex)=0;  virtual void insert(int theIndex,const T &x)=0;  virtual void output(ostream& out)const=0;  };  #include<algorithm>  #include<sstream>  #include<iterator>  template<class T>  class arrayList:public LinearList<T>  {  public:  arrayList(int initialCapacity);  arrayList(const arrayList<T>&);  arrayList()  {  element=new T [maxsize];  }  ~arrayList(){delete [] element;}  //重写父类虚构函数  virtual bool empty()const {return listSize==0;}  virtual int size()const {return listSize;}  virtual T get(int theIndex)const;  virtual int indexOf(const T&x)const;  virtual void erase(int theIndex);  virtual void insert(int theIndex,const T &x);  virtual void output(ostream& out)const;  //其他函数  int capacity()const{return arrayLength;}  void push\_back(const T&x);  void pop\_back();  void clear();  void set(int theIndex ,T theElement);  void reserve(const int& theCapacity);    protected:  void checkIndex(int theIndex)const;  T\* element;  int arrayLength=0;  int listSize=0;  };  template<class T>  void changeLength(T\*& a,int oldLength,int newLength)  {  if(newLength<0)return;  T \*temp=new T[newLength];  int size=min(oldLength,newLength);  copy(a,a+size,temp);  a=temp;  }  //直接构造函数  template<class T>  arrayList<T>::arrayList(int initialCapacity)  {  if(initialCapacity<1)return;  arrayLength=initialCapacity;  element=new T[arrayLength];  }  //拷贝函数  template<class T>  arrayList<T>::arrayList(const arrayList<T>& theList)  {  arrayLength= theList.arrayLength;  element=new T[arrayLength];  listSize=theList.listSize;;  copy(theList.element,theList.element+listSize,element);  }  //检查是否合法  template<class T>  void arrayList<T>::checkIndex(int theIndex)const  {  if(theIndex<0||theIndex>=listSize)  {  ostringstream s;  s<<"index="<<theIndex<<"size="<<listSize;  throw(s.str());  }  }  //返回元素  template<class T>  T arrayList<T>::get(int theIndex)const  {  checkIndex(theIndex);  return element[theIndex];  }  //找索引  template<class T>  int arrayList<T>::indexOf(const T&x)const  {  int theIndex=(int)(find(element,element+listSize,x)-element);  if(theIndex==listSize)  return -1;  else return theIndex;  }  //根据索引删除一个元素  template<class T>  void arrayList<T>::erase(int theIndex)  {  checkIndex(theIndex);  copy(element+theIndex+1,element+listSize,element+theIndex);  //释放最后一个元素的内存  element[--listSize].~T();  }  //插入函数  template<class T>  void arrayList<T>::insert(int theIndex,const T&x)  {  //检查是否为有效索引,可以写到末尾即listSize  if(theIndex<0||theIndex>listSize)  {  ostringstream s;  s<<"index="<<theIndex<<"size="<<listSize;  throw(s.str());  }  //满了则扩容  if(listSize==arrayLength)  {  changeLength(element,listSize,listSize\*2);  arrayLength\*=2;  }  //插入,往后移动  copy\_backward(element+theIndex,element+listSize,element+listSize+1);  element[theIndex]=x;  listSize++;  }  //输出  template<class T>  void arrayList<T>::output(ostream&out)const  {  if(listSize==0)  {  cout<<"empty";return;  }  //在元素之间插入空格  copy(element,element+listSize,ostream\_iterator<T>(out," "));  }  //重载  template<class T>  ostream& operator<<(ostream&out,const arrayList<T>&arr)  {  arr.output(out);return out;  }  //尾部插入一个元素  template<class T>  void arrayList<T>::push\_back(const T&x)  {  element[listSize++]=x;  }  //尾部删除一个元素  template<class T>  void arrayList<T>::pop\_back()  {  if(listSize<=0)  {  cout<<"already empty"<<endl;  return;  }  else  element[--listSize].~T();  }  //清空线性表  template<class T>  void arrayList<T>::clear()  {  listSize=0;  arrayLength=0;  }  template <class T>  inline void arrayList<T>::set(int theIndex, T theElement)  {  element[theIndex]=theElement;  listSize++;  }  template<class T>  void arrayList<T>::reserve(const int& theCapacity)  {  arrayLength = theCapacity;  listSize=0;  }  template<class T>  class MatrixTerms  {  public:  int row;  int col;  T value;  operator T()const {return value;}  };  template<class T>  class SparseMatrix  {  public:  SparseMatrix<T>(){};  SparseMatrix(int rows,int cols);  SparseMatrix(const SparseMatrix<T>&);  void transpose();  void operator\* (SparseMatrix<T>&b);  void operator+ (SparseMatrix<T>&b);  SparseMatrix<T>& operator= (SparseMatrix<T>&b);  void reSetSize(int rows,int cols);  void inputNonZero(int theNumberOfValues);  public:  int rows,cols;  arrayList<MatrixTerms<T>> terms;  };  //重载输出  template <class T>  ostream& operator<<(ostream& out, SparseMatrix<T>& x)  {  out<<x.rows<<" "<<x.cols<<endl;  int k=0;  for(int i=1;i<=x.rows;i++)  {  for(int j=1;j<=x.cols;j++)  {  bool flag=false;  if(k<x.terms.size())  {  auto tmp=x.terms.get(k);  if(tmp.row==i&&tmp.col==j)  {  cout<<tmp.value<<" ";  flag=true;  k++;  }  }  if(!flag)  cout<<"0"<<" ";  }  cout<<endl;  }  return out;  }  //重载输入  template<class T>  istream& operator>>(istream& in, SparseMatrix<T>& x)  {  MatrixTerms<T>mTerm;  int b;  //按行主列来存  for(int i=1;i<=x.rows;i++)  for(int j=1;j<=x.cols;j++)  {  in>>b;  if(b!=0)  {  mTerm.row=i;  mTerm.col=j;  mTerm.value=b;  x.terms.push\_back(mTerm);  }  }  return in;  }  //输入所有元素  //转置函数  template <class T>  inline void SparseMatrix<T>::transpose()  {  //只有转置矩阵的时候需要进行位置的选择  SparseMatrix<T>b;  b.reSetSize(cols,rows);  int colSize[cols+1];  int nextRow[cols+1];  for(int i=1;i<=cols;i++)  colSize[i]=0;  for(int i=0;i<terms.size();i++)  {  auto k=terms.get(i);  colSize[k.col]++;  }  nextRow[1]=0;  for(int i=2;i<=cols;i++)  nextRow[i]=nextRow[i-1]+colSize[i-1];  for(int i=0;i<terms.size();i++)  {  auto k=terms.get(i);  MatrixTerms<T>mTerm;  mTerm.row=k.col;  mTerm.col=k.row;  mTerm.value=k.value;    int index=nextRow[k.col]++;  b.terms.set(index,mTerm);  }  \*this=b;  };  template <class T>  void SparseMatrix<T>::operator+(SparseMatrix<T> &b)  {  if(rows!=b.rows||cols!=b.cols)  {  cout<<"-1"<<endl;  \*this=b;  return;  }  SparseMatrix<T>c;  c.reSetSize(rows,cols);  int i=0,j=0;  int n=terms.size();  int m=b.terms.size();  while(i<n&&j<m)  {  auto k1=terms.get(i);  auto k2=b.terms.get(j);    int idx1=k1.row\*cols+k1.col;  int idx2=k2.row\*cols+k2.col;  if(idx1<idx2)  {  c.terms.push\_back(k1);  i++;  }  else if(idx1==idx2)  {  MatrixTerms<T>mTerm;  mTerm.row=k1.row;  mTerm.col=k1.col;  mTerm.value=k1.value+k2.value;  c.terms.push\_back(mTerm);  i++,j++;  }  else  {  c.terms.push\_back(k2);  j++;  }  }  //补充剩下的  while(i<n)  {  c.terms.push\_back(terms.get(i));  i++;  }  while(j<m)  {  c.terms.push\_back(b.terms.get(j));  j++;  }    \*this=c;  }  template <class T>  inline void SparseMatrix<T>::operator\*(SparseMatrix<T> &b)  {  if(cols!=b.rows)  {  cout<<"-1"<<endl;  \*this=b;  return;  }  //结果矩阵  SparseMatrix<T>c;  c.reSetSize(rows,b.cols);  //思路:按左矩阵的每一行列举,找到相同行的非零元素,找到对应列,再通过找到右矩阵相同行上的元素进行累加即可  T rowSize[b.rows+1];  T nextRow[b.rows+1];  T answerRow[b.cols+1];  for(int i=1;i<=b.rows;i++)  rowSize[i]=0;  for(int i=0;i<b.terms.size();i++)  {  auto k=b.terms.get(i);  rowSize[k.row]++;  }  nextRow[1]=0;  for(int i=2;i<=b.rows;i++)  nextRow[i]=nextRow[i-1]+rowSize[i-1];  int p=0;  for(int i=1;p<terms.size()&&i<=rows;i++)  {  for(int k=1;k<=b.cols;k++)  answerRow[k]=0;  //左侧矩阵有该行元素,而且该元素右侧列有元素  while(p<terms.size()&&terms.get(p).row==i)  {  if(rowSize[terms.get(p).col]!=0)  {  auto k=terms.get(p);  for(int q=nextRow[k.col];q<nextRow[k.col]+rowSize[k.col];q++)  {  auto tmp=b.terms.get(q);  answerRow[tmp.col]+=k.value\*tmp.value;  }  }  p++;  }  //将answer输入  for(int j=1;j<=b.cols;j++)  {  if (answerRow[j]!=0)  {  MatrixTerms<T> mTerm;  mTerm.col=j;  mTerm.row=i;  mTerm.value=answerRow[j];  c.terms.push\_back(mTerm);  }  }  }  //把c赋给原矩阵  \*this=c;  }  template<class T>  SparseMatrix<T>& SparseMatrix<T>::operator=(SparseMatrix<T>&x)  {  reSetSize(x.rows,x.cols);  for(int i=0;i<x.terms.size();i++)  {  auto k=x.terms.get(i);  terms.push\_back(k);  }  return \*this;  }  template <class T>  inline SparseMatrix<T>::SparseMatrix(int rows,int cols)  {  this->rows=rows;  this->cols=cols;  }  template <class T>  inline SparseMatrix<T>::SparseMatrix(const SparseMatrix<T> &x)  {  \*this->reSetSize(x.rows,x.cols);  for(int i=0;i<x.terms.size();i++)  {  auto k=x.terms.get(i);  terms.push\_back(k);  }  }  template <class T>  inline void SparseMatrix<T>::reSetSize(int rows, int cols)  {  this->rows=rows;  this->cols=cols;  terms.reserve(maxsize);  }  template <class T>  inline void SparseMatrix<T>::inputNonZero(int theNumberOfValues)  {  MatrixTerms<T>mTerm;  for(int i=0;i<theNumberOfValues;i++)  {  scanf("%d%d%d",&mTerm.row,&mTerm.col,&mTerm.value);  terms.push\_back(mTerm);  }  }  int main()  { SparseMatrix<int>s;  int w,op;  scanf("%d",&w);  while(w--)  {  scanf("%d",&op);  switch (op)  {  case 1:  {  int n,m;  scanf("%d%d",&n,&m);  s.reSetSize(n,m);  cin>>s;  break;  }  case 2:  {  int n,m,t;  scanf("%d%d%d",&n,&m,&t);  //初始化右矩阵  SparseMatrix<int>tmp;  tmp.reSetSize(n,m);  tmp.inputNonZero(t);  s\*tmp;  break;  }  case 3:  {  int n,m,t;  scanf("%d%d%d",&n,&m,&t);  //初始化右矩阵  SparseMatrix<int>tmp;  tmp.reSetSize(n,m);  tmp.inputNonZero(t);  s+tmp;  break;  }  case 4:  {  cout<<s;  break;  }  case 5:  {  s.transpose();  break;  }  default:  break;  }  }  return 0;  } | | | |
|  | | | |