

稀疏矩阵的快速转置 (C语言) 算法详解

《[稀疏矩阵的转置算法](#)》一节介绍了实现矩阵转置的普通算法，该算法的[时间复杂度](#)为 $O(n^2)$ 。本节给大家介绍一种实现矩阵转置更高效的算法，通常称为[稀疏矩阵的快速转置算法](#)。

我们知道，稀疏矩阵的转置需要经历以下 3 步：

- 1. 将矩阵的行数和列数互换；
- 2. 将三元组表（存储矩阵）中的 i 列和 j 列互换，实现矩阵的转置；
- 3. 以 j 列为序，重新排列三元组表中存储各三元组的先后顺序；

稀疏矩阵快速转置算法和普通算法的区别仅在于第 3 步，快速转置能够做到遍历一次三元组表即可完成第 3 步的工作。

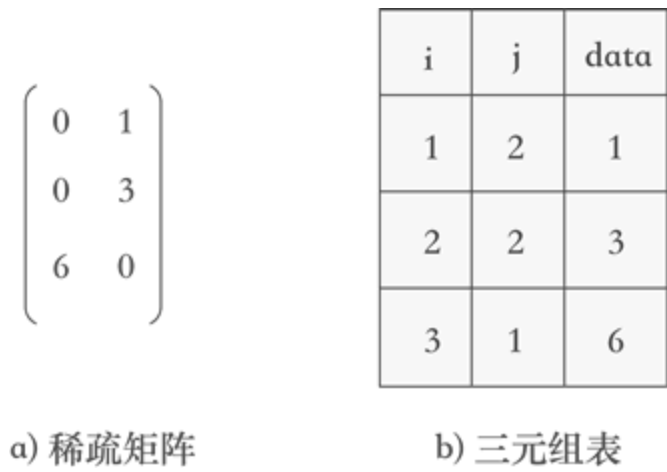


图 1 稀疏矩阵和对应的三元组表

如图 1 所示，此为转置之前的矩阵和对应的三元组表。稀疏矩阵的快速转置是这样的，在普通算法的基础上增设两个[数组](#)（假设分别为 array 和 copt）：

- array 数组负责记录原矩阵每一列非 0 元素的个数。以图 1 为例，则对应的array数组如图 2 所示：

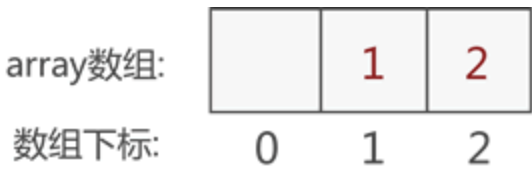


图 2 每一列非 0 元素的个数

图 2 中 array 数组表示，原稀疏矩阵中第一列有 1 个非 0 元素，第二列有 2 个非 0 元素。

- `cpot` 数组用于计算稀疏矩阵中每列第一个非 0 元素在新三元组表中存放的位置。我们通常默认第一列首个非 0 元素存放到新三元组表中的位置为 1，然后通过 `cpot[col] = cpot[col-1] + array[col-1]` 公式可计算出后续各列首个非 0 元素存放到新三元组表的位置。拿图 1 中的稀疏矩阵来说，它对应的 `cpot` 数组如图 3 所示：

cpot数组:		1	2
数组下标:	0	1	2

图 3 `cpot` 数组示意图

图 3 中的 `cpot` 数组表示，原稀疏矩阵中第 2 列首个非 0 元素存放到新三元组表的位置为 2。

注意，`cpot[col] = cpot[col-1] + array[col-1]` 的意思是，后一列首个非 0 元素存放的位置等于前一列首个非 0 元素的存放位置加上该列非 0 元素的个数。由此可以看出，`cpot` 数组才是最终想要的，而 `array` 数组的设立只是为了帮助我们得到 `cpot` 数组。

这样在实现第 3 步时，根据每个三元组中 `j` 的数值，可以借助 `cpot` 数组直接得到此三元组新的存放位置，C 语言实现代码如下：

```
01. //实现快速转置算法的函数
02. TSMatrix fastTransposeMatrix(TSMatrix M,TSMatrix T){
03.     //第1步：行和列置换
04.     T.m=M.n;
05.     T.n=M.m;
06.     T.num=M.num;
07.     if (T.num) {
08.         //计算array数组
09.         int array[number];
10.         for (int col=1; col<=M.m; col++) {
11.             array[col]=0;
12.         }
13.         for (int t=0; t<M.num; t++) {
14.             int j=M.data[t].j;
15.             array[j]++;
16.         }
17.
18.         //创建并初始化cpot数组
19.         int cpot[T.m+1];
20.         cpot[1]=1;//第一列中第一个非0元素的位置默认为1
21.         for (int col=2; col<=M.m; col++) {
22.             cpot[col]=cpot[col-1]+array[col-1];
23.         }
24.         //遍历一次即可实现三元组表的转置
25.         for (int p=0; p<M.num; p++) {
26.             //提取当前三元组的列数
27.             int col=M.data[p].j;
```

```

28.         //根据列数和cpot数组，找到当前元素需要存放的位置
29.         int q=cpot[col];
30.         //转置矩阵的三元组默认从数组下标0开始，而得到的q值是单纯的位置，所以要减1
31.         T.data[q-1].i=M.data[p].j;
32.         T.data[q-1].j=M.data[p].i;
33.         T.data[q-1].data=M.data[p].data;
34.         //存放完成后，cpot数组对应的位置要+1，以便下次该列存储下一个三元组
35.         cpot[col]++;
36.     }
37. }
38. return T;
39. }

```

使用 fastTransposeMatrix 函数实现图 1 中稀疏矩阵转置的 C 语言完整程序为：

```

01. #include<stdio.h>
02. #define number 10
03. typedef struct {
04.     int i,j;
05.     int data;
06. }triple;
07. typedef struct {
08.     triple data[number];
09.     int rpos[number];
10.     int n,m,num;
11. }TSMatrix;
12.
13. //fastTransposeMatrix放置位置
14.
15. int main() {
16.     TSMatrix M;
17.     M.m=2;
18.     M.n=3;
19.     M.num=3;
20.
21.     M.data[0].i=1;
22.     M.data[0].j=2;
23.     M.data[0].data=1;
24.
25.     M.data[1].i=2;
26.     M.data[1].j=2;
27.     M.data[1].data=3;
28.
29.     M.data[2].i=3;
30.     M.data[2].j=1;
31.     M.data[2].data=6;

```

```

32.
33.     TSMatrix T;
34.     T=fastTransposeMatrix(M, T);
35.     printf("转置矩阵三元组表为: \n");
36.     for (int i=0; i<T.num; i++) {
37.         printf("(%d,%d,%d)\n",T.data[i].i,T.data[i].j,T.data[i].data);
38.     }
39.     return 0;
40. }

```

程序运行结果为：

转置矩阵三元组表为：

(1,3,6)

(2,1,1)

(2,2,3)

可以看出，稀疏矩阵快速转置算法的时间复杂度为 $O(n)$ 。即使在最坏的情况下（矩阵中全部都是非 0 元素），该算法的时间复杂度也才为 $O(n^2)$ 。

联系方式 **购买教程（带答疑）**