**Experiment 2**

**AIM:** Write a program to multiply 2 sparse matrices using transpose.

**THEORY:** A matrix is a two-dimensional data object made of m rows and n columns, therefore having total m x n values. If most of the elements of the matrix have 0 value, then it is called a sparse matrix.  
Why to use Sparse Matrix instead of simple matrix?

* Storage: There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.
* Computing time: Computing time can be saved by logically designing a data structure traversing only non-zero elements.

**Code:**

1. #include < iostream >
2. #include < algorithm >
3. #include < vector >
4. using namespace std;
5. struct sparse {
6. int row, col, val;
7. sparse() {}
8. sparse(int r, int c, int v) {
9. row = r;
10. col = c;
11. val = v;
12. }
13. };
14. void print\_sparse(sparse \* s) {
15. int k = s[0].val;
16. for (int i = 0; i <= k; i++)
17. cout << s[i].row << " " << s[i].col << " " << s[i].val << endl;
18. cout << endl;
19. }
20. void print\_matrix(sparse \* s) {
21. int r = s[0].row, c = s[0].col, k = 1;
22. for (int i = 0; i < r; i++) {
23. for (int j = 0; j < c; j++) {
24. if (s[k].row == i && s[k].col == j)
25. cout << s[k++].val << " ";
26. else
27. cout << "0 ";
28. }
29. cout << endl;
30. }
31. }
32. sparse a[20], b[20], c[20];
33. void take\_input\_sparse(sparse \* s) {
34. int r, c, v;
35. cin >> r >> c >> v;
36. s[0] = sparse(r, c, v);
37. for (int i = 1; i <= s[0].val; i++) {
38. cin >> r >> c >> v;
39. s[i] = sparse(r, c, v);
40. }
41. }
42. //sort for transpose
43. bool cmp(sparse a, sparse b) {
44. if (a.row != b.row)
45. return a.row < b.row;
46. return a.col < b.col;
47. }
48. void transpose\_sparse(sparse \* s) {
49. //O(k\*log(k))
50. for (int i = 0; i <= s[0].val; i++)
51. swap(s[i].row, s[i].col);
52. sort(s + 1, s + 1 + s[0].val, cmp);
53. }
54. void multiply(sparse \* a, sparse \* b, sparse \* c) {
55. transpose\_sparse(b);
56. /\*
57. \* now we store the starting index
58. \* and ending index of every row (of the original matrix)
59. \* in the sparse matrix
60. \*/
61. //row#, lower index, upper index
62. vector < pair < int, pair < int, int > > > la, lb;
63. la.push\_back(make\_pair(a[1].row, make\_pair(1, 1)));
64. for (int i = 2; i <= a[0].val; i++) {
65. sparse current = a[i];
66. int current\_row = la[la.size() - 1].first;
67. if (current.row == current\_row)
68. la[la.size() - 1].second.second++;
69. else
70. la.push\_back(make\_pair(a[i].row, make\_pair(i, i)));
71. }
72. lb.push\_back(make\_pair(b[1].row, make\_pair(1, 1)));
73. for (int i = 2; i <= b[0].val; i++) {
74. sparse current = b[i];
75. int current\_row = lb[lb.size() - 1].first;
76. if (current.row == current\_row)
77. lb[lb.size() - 1].second.second++;
78. else
79. lb.push\_back(make\_pair(b[i].row, make\_pair(i, i)));
80. }
81. int k = 1;
82. /\* now multiply every row
83. \* of a with every row of b
84. \*/
85. for (auto i: la) {
86. for (auto j: lb) {
87. int cell = 0;
88. int ai = i.second.first, bj = j.second.first;
89. while (ai <= i.second.second && bj <= j.second.second) {
90. if (a[ai].col == b[bj].col)
91. cell += a[ai++].val \* b[bj++].val;
92. if (a[ai].col < b[bj].col)
93. ai++;
94. else
95. bj++;
96. }
97. if (cell)
98. c[k++] = sparse(i.first, j.first, cell);
99. }
100. }
101. c[0].row = a[0].row;
102. c[0].col = b[0].col;
103. c[0].val = k - 1;
104. }
105. int main() {
106. take\_input\_sparse(a);
107. take\_input\_sparse(b);
108. multiply(a, b, c);
109. cout << endl;
110. print\_sparse(c);
111. return 0;
112. }

**Output:**

