Indexed Storage of Sparse Matrix

Deddy Jobson

In science, one encounters many matrices which are of dimensions $n \times n$ but are mostly filled with zeros, save for about $O(n)$ elements. If no compression is done, multiplication of a matrix and a vector has a time complexity of $O(n^2)$. So certain methods have been devised to efficiently store such matrices to save space and also increasing the speed of certain operations like matrix multiplication.

0.0.1 Linked List Storage Method

Generally, even if the elements of a sparse matrix are mostly zero, it's diagonal elements are mostly non zero. Hence we can store all of the diagonal elements without much waste of space in a linked list. Then the non-diagonal non-zero elements are stored. In a new linked list, the first n elements store the occurrence of the first non-diagonal non-zero elements in each row in the previous linked list. The next element stores the index after the last element in the last row. Finally, the column number of all the corresponding elements in the previous linked list is stored.

This compressed form of the matrix stores at most $2(n+1+k)$ terms where k is the number of non-zero non-diagonal elements. As k is of $O(n)$, the total space complexity is also of $O(n)$. The main advantages, however, can be seen during matrix multiplication.

When the compressed matrix is multiplied with a vector, all the diagonal elements (n terms) are multiplied with the corresponding term in the vector. Then, the non-zero non-diagonal elements (k terms) are multiplied with the vector term which corresponds to it's column number which is stored in the other linked list.
So the time complexity T can be calculated by

$$\begin{aligned} T &= O(n) + O(k) \\ &= O(n) + O(O(n)) \\ T &= O(n) \end{aligned}$$

Hence even the time complexity of matrix multiplication with a vector has been improved from $O(n^2)$ to $O(n)$. This improvement is present also for multiplication of a vector's transpose with a matrix.

0.0.2 Arrays vs Linked List

This form of compression can be implemented using arrays or using linked list. The disadvantage of static array implementation is that the number of elements must be guessed which leads to either an array which is too small or a wastage of space. However, even though the time complexity is the same as that of linked list, the matrix multiplication is significantly faster using array. This has been observed by comparing the run times of both the functions with the given input.

For array,

$$t = 0.011ms$$

For linked list,

$$t = 0.021ms$$

A third alternative is to use dynamic arrays (also called vectors in C++). It is an array whose size grows whenever the array's limit is reached. This ensures that the amount of space wasted does not exceed a certain percentage and also allows us to do matrix multiplication as fast as the array implementation. The only disadvantage is that to compress a matrix, using dynamic arrays takes more time than static arrays, perhaps even more than linked list. The greater it's growth factor, the lesser the space wastage and greater the time taken to compress the matrix.

Therefore, to summarize, there are 3 ways to implement the compression of the matrix.

- Linked List:Used if you want no wastage and don't mind processes being a bit slower.
- Static Array:Used if you don't mind wasting memory and want maximum speed.
- Dynamic Array: It is a compromise between the above 2 methods.