Sparse Matrix Multiplication

Given two sparse matrices **A** and **B**, return the result of **AB**.

You may assume that **A**'s column number is equal to **B**'s row number.

Example:

```
A = [
  [ 1, 0, 0],
  [-1, 0, 3]
]

B = [
  [ 7, 0, 0 ],
  [ 0, 0, 0 ],
  [ 0, 0, 1 ]
]


     |  1 0 0 |   | 7 0 0 |   |  7 0 0 |
AB = | -1 0 3 | x | 0 0 0 | = | -7 0 3 |
                  | 0 0 1 |
```

Solution 1

UPDATE: Thanks to @stpeterh we have this `70ms` concise solution:

```
public class Solution {
    public int[][] multiply(int[][] A, int[][] B) {
        int m = A.length, n = A[0].length, nB = B[0].length;
        int[][] C = new int[m][nB];

        for(int i = 0; i < m; i++) {
            for(int k = 0; k < n; k++) {
                if (A[i][k] != 0) {
                    for (int j = 0; j < nB; j++) {
                        if (B[k][j] != 0) C[i][j] += A[i][k] * B[k][j];
                    }
                }
            }
        }
        return C;
    }
}
```

The followings is the original `75ms` solution:

The idea is derived from a CMU lecture.

> A sparse matrix can be represented as a sequence of rows, each of which is a sequence of (column-number, value) pairs of the nonzero values in the row.

So let's create a non-zero array for A, and do multiplication on B.

Hope it helps!

```java
public int[][] multiply(int[][] A, int[][] B) {
    int m = A.length, n = A[0].length, nB = B[0].length;
    int[][] result = new int[m][nB];

    List[] indexA = new List[m];
    for(int i = 0; i < m; i++) {
        List<Integer> numsA = new ArrayList<>();
        for(int j = 0; j < n; j++) {
            if(A[i][j] != 0){
                numsA.add(j);
                numsA.add(A[i][j]);
            }
        }
        indexA[i] = numsA;
    }

    for(int i = 0; i < m; i++) {
        List<Integer> numsA = indexA[i];
        for(int p = 0; p < numsA.size() - 1; p += 2) {
            int colA = numsA.get(p);
            int valA = numsA.get(p + 1);
            for(int j = 0; j < nB; j ++) {
                int valB = B[colA][j];
                result[i][j] += valA * valB;
            }
        }
    }

    return result;
}
```

written by yavinci original link here

Solution 2

Given A and B are sparse matrices, we could use lookup tables to speed up. At the begining I thought two lookup tables would be necessary. After discussing with @yavinci, I think one lookup table for B would be enough. Surprisingly, it seems like detecting non-zero elements for both A and B on the fly without additional data structures provided the fastest performance on current test set.

However, I think such fastest performance could due to an imperfect test set we have for OJ right now: there are only 12 test cases. And, for an element `B[k, j]`, it would be detected for non-zero elements several times if we detecting both A and B on the fly, depending on how many `i`'s make elements `A[i, k]` non-zero. With this point, the additional data structures, like lookup tables, should save our time by focusing on only non-zero elements. If it is not, I am worried the set of OJ test cases probably is not good enough.

Anyway, I am posting my respective solutions below. Comments are welcome. Thanks @yavinci again for discussing with me.

Python solution with only one table for B (~196ms):

```python
class Solution(object):
    def multiply(self, A, B):
        """
        :type A: List[List[int]]
        :type B: List[List[int]]
        :rtype: List[List[int]]
        """
        if A is None or B is None: return None
        m, n, l = len(A), len(A[0]), len(B[0])
        if len(B) != n:
            raise Exception("A's column number must be equal to B's row number.")
        C = [[0 for _ in range(l)] for _ in range(m)]
        tableB = {}
        for k, row in enumerate(B):
            tableB[k] = {}
            for j, eleB in enumerate(row):
                if eleB: tableB[k][j] = eleB
        for i, row in enumerate(A):
            for k, eleA in enumerate(row):
                if eleA:
                    for j, eleB in tableB[k].iteritems():
                        C[i][j] += eleA * eleB
        return C
```

Java solution with only one table for B (~150ms):

```java
public class Solution {
    public int[][] multiply(int[][] A, int[][] B) {
        if (A == null || A[0] == null || B == null || B[0] == null) return null;
        int m = A.length, n = A[0].length, l = B[0].length;
        int[][] C = new int[m][l];
        Map<Integer, HashMap<Integer, Integer>> tableB = new HashMap<>();

        for(int k = 0; k < n; k++) {
            tableB.put(k, new HashMap<Integer, Integer>());
            for(int j = 0; j < l; j++) {
                if (B[k][j] != 0){
                    tableB.get(k).put(j, B[k][j]);
                }
            }
        }

        for(int i = 0; i < m; i++) {
            for(int k = 0; k < n; k++) {
                if (A[i][k] != 0){
                    for (Integer j: tableB.get(k).keySet()) {
                        C[i][j] += A[i][k] * tableB.get(k).get(j);
                    }
                }
            }
        }
        return C;
    }
}
```

Python solution without table (~156ms):

```python
class Solution(object):
    def multiply(self, A, B):
        """
        :type A: List[List[int]]
        :type B: List[List[int]]
        :rtype: List[List[int]]
        """
        if A is None or B is None: return None
        m, n, l = len(A), len(A[0]), len(B[0])
        if len(B) != n:
            raise Exception("A's column number must be equal to B's row number.")
        C = [[0 for _ in range(l)] for _ in range(m)]
        for i, row in enumerate(A):
            for k, eleA in enumerate(row):
                if eleA:
                    for j, eleB in enumerate(B[k]):
                        if eleB: C[i][j] += eleA * eleB
        return C
```

Java solution without table (~70ms):

```java
public class Solution {
    public int[][] multiply(int[][] A, int[][] B) {
        int m = A.length, n = A[0].length, l = B[0].length;
        int[][] C = new int[m][l];

        for(int i = 0; i < m; i++) {
            for(int k = 0; k < n; k++) {
                if (A[i][k] != 0){
                    for (int j = 0; j < l; j++) {
                        if (B[k][j] != 0) C[i][j] += A[i][k] * B[k][j];
                    }
                }
            }
        }
        return C;
    }
}
```

Python solution with two tables (~196ms):

```python
class Solution(object):
    def multiply(self, A, B):
        """
        :type A: List[List[int]]
        :type B: List[List[int]]
        :rtype: List[List[int]]
        """
        if A is None or B is None: return None
        m, n = len(A), len(A[0])
        if len(B) != n:
            raise Exception("A's column number must be equal to B's row number.")
        l = len(B[0])
        table_A, table_B = {}, {}
        for i, row in enumerate(A):
            for j, ele in enumerate(row):
                if ele:
                    if i not in table_A: table_A[i] = {}
                    table_A[i][j] = ele
        for i, row in enumerate(B):
            for j, ele in enumerate(row):
                if ele:
                    if i not in table_B: table_B[i] = {}
                    table_B[i][j] = ele
        C = [[0 for j in range(l)] for i in range(m)]
        for i in table_A:
            for k in table_A[i]:
                if k not in table_B: continue
                for j in table_B[k]:
                    C[i][j] += table_A[i][k] * table_B[k][j]
        return C
```

Java solution with two tables (~160ms):

```
public class Solution {
    public int[][] multiply(int[][] A, int[][] B) {
        if (A == null || B == null) return null;
        if (A[0].length != B.length)
            throw new IllegalArgumentException("A's column number must be equal t
o B's row number.");
        Map<Integer, HashMap<Integer, Integer>> tableA = new HashMap<>();
        Map<Integer, HashMap<Integer, Integer>> tableB = new HashMap<>();
        int[][] C = new int[A.length][B[0].length];
        for (int i = 0; i < A.length; i++) {
            for (int j = 0; j < A[i].length; j++) {
                if (A[i][j] != 0) {
                    if(tableA.get(i) == null) tableA.put(i, new HashMap<Integer,
Integer>());
                    tableA.get(i).put(j, A[i][j]);
                }
            }
        }

        for (int i = 0; i < B.length; i++) {
            for (int j = 0; j < B[i].length; j++) {
                if (B[i][j] != 0) {
                    if(tableB.get(i) == null) tableB.put(i, new HashMap<Integer,
Integer>());
                    tableB.get(i).put(j, B[i][j]);
                }
            }
        }

        for (Integer i: tableA.keySet()) {
            for (Integer k: tableA.get(i).keySet()) {
                if (!tableB.containsKey(k)) continue;
                for (Integer j: tableB.get(k).keySet()) {
                    C[i][j] += tableA.get(i).get(k) * tableB.get(k).get(j);
                }
            }
        }
        return C;
    }
}
```

written by stpeterh original link here

Solution 3

```cpp
class Solution {
public:
    vector<vector<int>> multiply(vector<vector<int>>& A, vector<vector<int>>& B)
{
        int m = A.size(), n = A[0].size();
        vector<vector<int>> res(m, vector<int>(B[0].size(),0));

        // for(int i = 0; i < m; i++){
        //     for(int k = 0; k < n; k++){
        //          for(int j = 0; j < n; j++){
        //              res[i][j] += A[i][k] * B[k][j];
        //          }
        //     }
        // }
        // imporved on upper version, this is a math solution
        for(int i = 0; i < m; i++){
            for(int k = 0; k < n; k++){
                if(A[i][k] != 0)
                    for(int j = 0; j < B[0].size(); j++){
                        res[i][j] += A[i][k] * B[k][j];
                    }
            }
        }
        return res;
    }
};
```

written by qilong.ma.3 original link here