Design Excel Sum Formula

Your task is to design the basic function of Excel and implement the function of sum formula. Specifically, you need to implement the following functions:

`Excel(int H, char W):` This is the constructor. The inputs represents the height and width of the Excel form. **H** is a positive integer, range from 1 to 26. It represents the height. **W** is a character range from 'A' to 'Z'. It represents that the width is the number of characters from 'A' to **W**. The Excel form content is represented by a height * width 2D integer array `C`, it should be initialized to zero. You should assume that the first row of `C` starts from 1, and the first column of `C` starts from 'A'.

`void Set(int row, char column, int val):` Change the value at `C(row, column)` to be val.

`int Get(int row, char column):` Return the value at `C(row, column)`.

`int Sum(int row, char column, List of Strings : numbers):` This function calculate and set the value at `C(row, column)`, where the value should be the sum of cells represented by `numbers`. This function return the sum result at `C(row, column)`. This sum formula should exist until this cell is overlapped by another value or another sum formula.

`numbers` is a list of strings that each string represent a cell or a range of cells. If the string represent a single cell, then it has the following format : `ColRow`. For example, "F7" represents the cell at (7, F).

If the string represent a range of cells, then it has the following format : `ColRow1:ColRow2`. The range will always be a rectangle, and ColRow1 represent the position of the top-left cell, and ColRow2 represents the position of the bottom-right cell.

Example 1:

```
Excel(3,"C");
// construct a 3*3 2D array with all zero.
//   A B C
// 1 0 0 0
// 2 0 0 0
// 3 0 0 0

Set(1, "A", 2);
// set C(1,"A") to be 2.
//   A B C
// 1 2 0 0
// 2 0 0 0
// 3 0 0 0

Sum(3, "C", ["A1", "A1:B2"]);
// set C(3,"C") to be the sum of value at C(1,"A") and the values sum of the rectangl
e range whose top-left cell is C(1,"A") and bottom-right cell is C(2,"B"). Return 4.
//   A B C
// 1 2 0 0
// 2 0 0 0
// 3 0 0 4

Set(2, "B", 2);
// set C(2,"B") to be 2. Note C(3, "C") should also be changed.
//   A B C
// 1 2 0 0
// 2 0 2 0
// 3 0 0 6
```

Note:

1. You could assume that there won't be any circular sum reference. For example, A1 = sum(B1) and B1 = sum(A1).
2. The test cases are using double-quotes to represent a character.
3. Please remember to **RESET** your class variables declared in class Excel, as static/class variables are **persisted across multiple test cases**. Please see here for more details.

Solution 1

I have put detailed explanation in the comments. A vector<vector<int>> is used for the Excel. Here I just want to highlight two things.

1. When a cell changes, all the sum related have to update. So a forward link from a cell to all the cells with it in the sum is required. Because a cell may be used for multiple times in the sum due to overlapping ranges, unordered_map<cell, unordered_map<cell, weight>> is used. The weight could improve the efficiency in the worst case.
2. When reset a cell's value, or reassign a new sum range to it, the cells contribute to its sum will change. So a backward link from this cell to all those cells is necessary. unordered_map<cell, unordered_set<cell>> is sufficient.

The excel is small, at most 26 by 26. So we can use row*26+col as the key of a cell, which is int.
See the code below.

```
class Excel {
public:
    Excel(int H, char W) {
        // initialization. Because r starts from 1, I used H+1 instead of H.
        Exl = vector<vector<int>> (H+1, vector<int>(W-'A'+1, 0));
        fward.clear();
        bward.clear();
    }

    void set(int r, char c, int v) {
        int col = c-'A', key = r*26+col;
        // update its value and all the sum related
        update(r, col, v);
        // This is a reset, so break all the forward links if existing
        if (bward.count(key)) {
            for (int k:bward[key]) {
                fward[k].erase(key);
            }
            bward[key].clear();
        }
    }
    // update a cell and all the sum related, using BFS
    // weights are used to improve efficiency for overlapping ranges
    void update(int r, int col, int v) {
        int prev = Exl[r][col];
        Exl[r][col] = v;
        // myq is keys for cells in next level, and update is the increment for each
cell
        queue<int> myq, update;
        myq.push(r*26+col);
        update.push(v-prev);
        while (!myq.empty()) {
            int key = myq.front(), dif = update.front();
            myq.pop();
            update.pop();
            if (fward.count(key)) {
                for (auto it = fward[key].begin(); it != fward[key].end(); it++) {
                    int k = it->first;
                    myq.push(k);
                    update.push(dif*(it->second));
                    Exl[k/26][k%26] += dif*(it->second);
                }
            }
        }
    }

    int get(int r, char c) {
        return Exl[r][c-'A'];
    }

    int sum(int r, char c, vector<string> strs) {
        // this is another reset, so break all the forward links
        int col = c-'A', key = r*26+col, ans = 0;
        if (bward.count(key)) {
            for (int k:bward[key]) {
                fward[k].erase(key);
            }
            bward[key].clear();
        }
        // get the sum over multiple ranges
        for (string str:strs) {
            int p = str.find(':'), left, right, top, bot;
            left = str[0]-'A';
            right = str[p+1]-'A';
            if (p == -1)
                top = stoi(str.substr(1));
            else
                top = stoi(str.substr(1, p-1));
            bot = stoi(str.substr(p+2));
            for (int i = top; i <= bot; ++i) {
                for (int j = left; j <= right; ++j) {
                    ans += Exl[i][j];
                    fward[i*26+j][key]++;
                    bward[key].insert(i*26+j);
                }
            }
        }
        // update this cell and all the sum related
        update(r, col, ans);
        return ans;
    }
private:
    // The key of a cell is defined as 26*row+col, which is int;
    // fward links a cell to all the cells which use it for sum, and it may be used
for
    // multiple times due to overlap ranges, so another map is used for its weight
    // bward links a cell to all the cells that contribute to its sum. When reset it
s value,
    // or reassign a new sum range to it, we need disconnect the forward link of tho
se cells to it.
    unordered_map<int, unordered_map<int, int>> fward;
    unordered_map<int, unordered_set<int>> bward;
    vector<vector<int>> Exl;
```

```
};
```

written by zestypanda original link here

Solution 2

```
class Excel {
private:
    int **dict;
    int offset, H, W;
    unordered_map<int, vector<string>> mp;
public:
    Excel(int H, char W) {
        offset = 'A';
        this->H = H;
        this->W = W - offset + 1;
        mp.clear();
        dict = new int*[H];
        for (int i = 0; i < H; i++) {
            dict[i] = new int[this->W];
            memset(dict[i], 0, sizeof(int)*this->W);
        }
    }

    void set (int r, char c, int v) {
        int k = (r << 10) + c;
        dict[r - 1][c - offset] = v;
        mp.erase(k);
    }

    int get (int r, char c) {
        int k = (r << 10) + c;
        if (mp.find(k) == mp.end())
            return dict[r - 1][c - offset];
        return get_cells(mp[k]);
    }

    int sum (int r, char c, vector<string> strs) {
        int k = (r << 10) + c;
        dict[r - 1][c - offset] = get_cells(strs);
        mp[k] = strs;
        return dict[r - 1][c - offset];
    }

    int get_cells(vector<string> &strs) {
        int res = 0;
        for (auto s : strs) {
            if (s.find(':')  == -1)
                res += get_cell(s);
            else
                res += get_cell_range(s);
        }
        return res;
    }

    int get_cell(string &cell) {
        int r = 0, idx = 0;
        char c = cell[idx++];
        while (idx < cell.length())
            r = 10 * r + cell[idx++] - '0';
        return get(r, c);
        return get(r, c);
    }

    int get_cell_range(string &cell_range) {
        int rs = 0, re = 0, idx = 0, res = 0;
        char cs, ce;

        int seg = cell_range.find(':');
        cs = cell_range[idx++];
        while (idx < seg)
            rs = 10 * rs + cell_range[idx++] - '0';

        idx++;
        ce = cell_range[idx++];
        while (idx < cell_range.length())
            re = 10 * re + cell_range[idx++] - '0';

        for (int r = rs; r <= re; r++) {
            for (char c = cs; c <= ce; c++) {
                res += get(r, c);
            }
        }
        return res;
    }
};

/**
 * Your Excel object will be instantiated and called as such:
 * Excel obj = new Excel(H, W);
 * obj.set(r,c,v);
 * int param_2 = obj.get(r,c);
 * int param_3 = obj.sum(r,c,strs);
 */
```

written by jordandong original link here

Solution 3

```
class Excel(object):

    def __init__(self, H, W):
        self.M = [[{'v': 0, 'sum': None} for i in range(H)] for j in range(ord(W) -
64)]

    def set(self, r, c, v):
        self.M[r - 1][ord(c) - 65] = {'v': v, 'sum': None}

    def get(self, r, c):
        cell = self.M[r - 1][ord(c) - 65]
        if not cell['sum']: return cell['v']
        return sum(self.get(*pos) * cell['sum'][pos] for pos in cell['sum'])

    def sum(self, r, c, strs):
        self.M[r - 1][ord(c) - 65]['sum'] = self.parse(strs)
        return self.get(r, c)

    def parse(self, strs):
        c = collections.Counter()
        for s in strs:
            s, e = s.split(':')[0], s.split(':')[1] if ':' in s else s
            for i in range(int(s[1:]), int(e[1:]) + 1):
                for j in range(ord(s[0]) - 64, ord(e[0]) - 64 + 1):
                    c[(i, chr(j + 64))] += 1
        return c
```

written by lee215 original link here