Candy Crush

This question is about implementing a basic elimination algorithm for Candy Crush.

Given a 2D integer array `board` representing the grid of candy, different positive integers `board[i][j]` represent different types of candies. A value of `board[i][j] = 0` represents that the cell at position `(i, j)` is empty. The given board represents the state of the game following the player's move. Now, you need to restore the board to a *stable state* by crushing candies according to the following rules:

1. If three or more candies of the same type are adjacent vertically or horizontally, "crush" them all at the same time - these positions become empty.
2. After crushing all candies simultaneously, if an empty space on the board has candies on top of itself, then these candies will drop until they hit a candy or bottom at the same time. (No new candies will drop outside the top boundary.)
3. After the above steps, there may exist more candies that can be crushed. If so, you need to repeat the above steps.
4. If there does not exist more candies that can be crushed (ie. the board is *stable*), then return the current board.

You need to perform the above rules until the board becomes stable, then return the current board.

Example 1:

```
Input:
board =
[[110,5,112,113,114],[210,211,5,213,214],[310,311,3,313,314],[410,411,412,5,414],[5,1
,512,3,3],[610,4,1,613,614],[710,1,2,713,714],[810,1,2,1,1],[1,1,2,2,2],[4,1,4,4,1014
]]
Output:
[[0,0,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[110,0,0,0,114],[210,0,0,0,214],[310,0,0,113,314
],[410,0,0,213,414],[610,211,112,313,614],[710,311,412,613,714],[810,411,512,713,1014
]]
Explanation:
```

Note:

1. The length of `board` will be in the range [3, 50].
2. The length of `board[i]` will be in the range [3, 50].
3. Each `board[i][j]` will initially start as an integer in the range [1, 2000].

Solution 1

The idea is not to count how many same "candies" are in a row or column, but to check if this candy is eligible for crushing. If any candy is eligible, store the corresponding coordinates in a HashSet.
After traversing the whole board, set the valid candies to "0" then crush (using 2-pointer method in a column).
Here goes the code:

```
class Solution {
    public int[][] candyCrush(int[][] board) {
        Set<Coordinates> set = new HashSet<>();
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[i].length; j++) {
                int cur = board[i][j];
                if (cur == 0) continue;
                if ((i - 2 >= 0 && board[i - 1][j] == cur && board[i - 2][j] == cur
) ||                                                    // check left 2 candies
                   (i + 2 <= board.length - 1 && board[i + 1][j] == cur && board[i
+ 2][j] == cur) ||                                      // check right 2 candies
                   (j - 2 >= 0 && board[i][j - 1] == cur && board[i][j - 2] == cur)
||                                                    // check 2 candies top
                   (j + 2 <= board[i].length - 1 && board[i][j + 1] == cur && board
[i][j + 2] == cur) ||                                   // check 2 candies below
                   (i - 1 >= 0 && i + 1 <= board.length - 1 && board[i - 1][j] == c
ur && board[i + 1][j] == cur) ||                        // check if it is a mid candy i
n row
                   (j - 1 >= 0 && j + 1 <= board[i].length - 1 && board[i][j - 1] =
= cur && board[i][j + 1] == cur)) {                   // check if it is a mid candy in
column
                    set.add(new Coordinates(i, j));
                }
            }
        }
        if (set.isEmpty()) return board;      //stable board
        for (Coordinates c : set) {
            int x = c.i;
            int y = c.j;
            board[x][y] = 0;      // change to "0"
        }
        drop(board);
        return candyCrush(board);
    }
    private void drop(int[][] board) {                                          //
using 2-pointer to "drop"
        for (int j = 0; j < board[0].length; j++) {
            int bot = board.length - 1;
            int top = board.length - 1;
            while (top >= 0) {
                if (board[top][j] == 0) {
                    top--;
                }
                else {
                    board[bot--][j] = board[top--][j];
                }
            }
```

```
            while (bot >= 0) {
                board[bot--][j] = 0;
            }
        }
    }
}

class Coordinates {
    int i;
    int j;
    Coordinates(int x, int y) {
        i = x;
        j = y;
    }
}
```

written by jun1013 original link here

Solution 2

```
class Solution {
public:
    vector<vector<int>> candyCrush(vector<vector<int>>& b) {
        int n = b.size(), m = b[0].size();
        while (true) {
            vector<pair<int,int>> vp;
            for (int i = 0; i < n; ++i) for (int j = 0; j < m; ++j) if (b[i][j]) {
                int i0 = i, i1 = i, j0 = j, j1 = j;
                while (i1 < n &&  i1 < i + 3 && b[i1][j] == b[i][j]) ++i1;
                while (i0 >= 0 && i0 > i - 3 && b[i0][j] == b[i][j]) --i0;
                while (j1 < m && j1 < j + 3 && b[i][j1] == b[i][j]) ++j1;
                while (j0 >= 0 && j0 > j - 3 && b[i][j0] == b[i][j]) --j0;
                if (i1 - i0 >3 || j1 -j0 > 3) vp.push_back({i,j});
            }
            if (vp.empty()) break;
            for (auto p:vp) b[p.first][p.second] = 0;
            for (int j = 0; j < m; ++j) {
                int t = n-1;
                for (int i = n-1; i >=0; --i) if (b[i][j]) swap(b[i][j],b[t--][j]);
                for (int i = t; i >=0; --i) b[i][j] = 0;
            }
        }

        return b;
    }
};
```

written by elastico original link here

Solution 3

In the textual description of Example 1 the 2nd last line of the board is `[1,2,1,2,2]`
but the illustration says
`[1,1,2,2,2]`

So this is something to pay attention to maybe ...

Here's my simple & naive solution -- one function marks & counts the cells to be removed, and another function "evolves" the state of the board. The "Evolve" function is kind like "Move Zeroes". Not super duper fast but seemed to work ...

```
class Solution {
public:
 int Mark(vector<vector<int>> & board, vector<vector<bool>>* marked) {
  const int H = int(board.size()), W = int(board.back().size());
  // Horiz
  for (int y=0; y<H; y++) {
   int x=0;
   while (x < W) {
    int this_cell = board[y][x];
    int x1 = x;
    for (x1 = x; x1 < W; x1++) {
     if (this_cell == board[y][x1]) { }
     else break;
    }
    if (this_cell != 0 && x1 - x >= 3) {
     for (int xx=x; xx<x1; xx++) (*marked)[y][xx] = true;
    }
    x = x1;
   }
  }

  // Vert
  for (int x=0; x<W; x++) {
   int y=0;
   while (y < H) {
    int this_cell = board[y][x];
    int y1 = y;
    for (y1 = y; y1 < H; y1++) {
     if (this_cell == board[y1][x]) { } else break;
    }
    if (this_cell != 0 && y1 - y >= 3) {
     for (int yy=y; yy<y1; yy++) (*marked)[yy][x] = true;
    }
    y = y1;
   }
  }

  int ret = 0;
  for (int y=0; y<H; y++) {
   for (int x=0; x<W; x++) {
    ret += ((*marked)[y][x] == true);
   }
  }
```

```
  return ret;
 }

 void Evolve(vector<vector<int>>& board, vector<vector<bool>>* marked) {
  const int H = int(board.size()), W = int(board.back().size());
  for (int x=0; x<W; x++) {
   int y1 = H-1, y0 = H-1;
   while (y1 >= 0) {
    if ((*marked)[y1][x] == true) {
     y1 --;
    } else {
     board[y0][x] = board[y1][x];
     y1--; y0--;
    }
   }
   for (; y0 >= 0; y0--) board[y0][x] = 0;
  }
 }

    vector<vector<int>> candyCrush(vector<vector<int>>& board) {
  const int H = int(board.size()), W = int(board.back().size());
  vector<vector<bool> > marked(H, vector<bool>(W, false) );
  while (true) {
   int nm = Mark(board, &marked);
   if (nm > 0) { Evolve(board, &marked); } else { break; }
   for (int y=0; y<H; y++) for (int x=0; x<W; x++) marked[y][x] = false;
  }
  return board;
    }
};
```

written by quadpixels original link here