Bee Problem Solution

COMP321

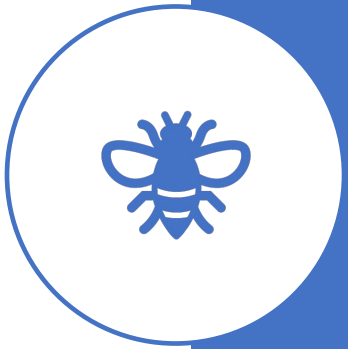
Team 6 Puffin

Christopher Zheng, Nadim Rahman, Maya Weinstein

Video Link: https://www.youtube.com/watch?v=ZRzsAJ34qls

Section 1: Key Information

- Bee storing honey in cells of a beehive
- Cells are either filled with old honey or empty
- Honey will continue to flow into empty adjacent cells
- Strategy: Pour honey into cells with many adjacent empty cells
- Goal: Do minimum amount of work to get rid of the honey you are carrying



I/O

Input

- 1st Row: *h n m*
  - *h:* Amount of honey to be stored, $0 \leq h \leq 10^6$
  - *n, m:* Dimensions of grid, $1 \leq n, m \leq 10^3$
- Following *n* lines:
  - *m* symbols separated by spaces, either:
    - " . " → empty cell
    - " # " → filled cell
  - Every other row starts with a space → slightly offset

Output

- Single integer: # of cells honey is directly funneled into to store all honey the bee is carrying

Sample Input 1

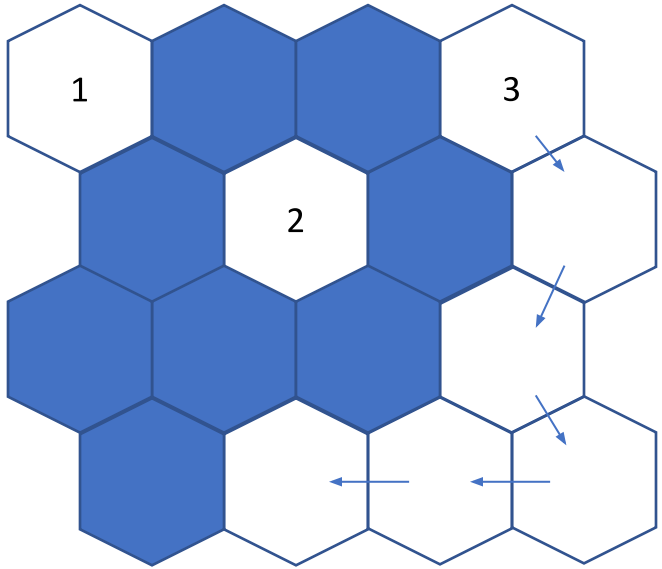
```
8 4 4
. # # .
 # . # .
# # # .
 # . . .
```

Sample Output 1

```
3
```

*Grid always has enough empty spaces to store all honey

*Number of empty cells can be greater than *h*



Section 2: Problem Analysis

Two things to bear in mind:

1. Maximum connected blocks
   a. Get and sort
   b. Different directions (->different solutions)

2. Input / Output

Initialization

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
#include<iostream>
#include<algorithm>

using namespace std;


// initialization
int counter;
int h, n, m;
int number;

const int N = 1e3 + 100;

// cell_map stores the configuration of the bee's cells.
char cell_map[N][N];
int answer[N * N];



//odd case
const int cell_odd[6][2]={0, 1, 0, -1, 1, 0, -1, 0, -1, -1, 1, -1};
//even case
const int cell_even[6][2]={0, 1, 0, -1, 1, 0, -1, 0, 1, 1, -1, 1};

```

Helpers

```
30  // helper function to compare two answers.
31  bool is_greater_than(int a,int b){
32      return a > b;
33  }
```

```
69  void output(){
70      int temp = 0;
71      int temp2 = 0;
72      int c = 0;
73      while(temp2 < h && c < counter){
74          temp += 1;
75          temp2 += answer[c];
76          c += 1;
77      }
78      //output
79      cout << temp << endl;
80  }
81
```

DFS

```
35  // We use DFS to traverse the whole map.
36  void DFS(int x,int y){
37      // we reach this point x,y so we fill this cell.
38      cell_map[x][y]='#';
39      // we increment the number of cells that we have traversed.
40      number = number + 1;
41
42      // we traverse along with six directions of a cell
43      // note the difference between odd and even rows of cells.
44      for(int i = 0; i < 6; i++){
45
46          int aa, bb;
47
48          // odd case
49          if(x&1){
50              aa = cell_even[i][0] + x;
51              bb = cell_even[i][1] + y;
52          }// even case
53          else{
54              aa = cell_odd[i][0] + x;
55              bb = cell_odd[i][1] + y;
56          }
57
58          // check if this cell is filled already.
59          if(cell_map[aa][bb]=='#') continue;
60          // or if a cell is out of range (no harm in this case)
61          if(aa<0 || aa>=n || bb<0 || bb>=m) continue;
62
63          // we need to keep updating during the DFS.
64          // if we pass all the checks, we head into DFS again starting at this cell.
65          DFS(aa,bb);
66      }
67  }
```

Main method

```
85  int main(){
86      // read the first line of three numbers.
87      int unused_val = scanf("%d%d%d",&h,&n,&m);
88      counter=0;
89
90      getchar(); // important
91
92      // read and store the whole configuration
93      for(int i = 0; i < n; i++){
94          for(int j = 0; j < m; j++){
95              cin >> cell_map[i][j];
96          }
97      }
98
99      // traverse the whole configuration by looping and DFS
100     for(int i = 0; i < n; i++){
101         for(int j = 0; j < m; j++){
102             if(cell_map[i][j] == '.'){
103                 // for each starting cell, we init a new number
104                 // to store the number of cells traversed.
105                 number = 0;
106                 DFS(i,j);
107                 answer[counter] = number;
108                 counter += 1;
109             }
110         }
111     }
112
113     // compare and sort the connected blocks.
114     sort(answer, answer+counter, is_greater_than);
115
116     output();
117     return 0;
118 }
```

Alternative Java Solution

Taking input

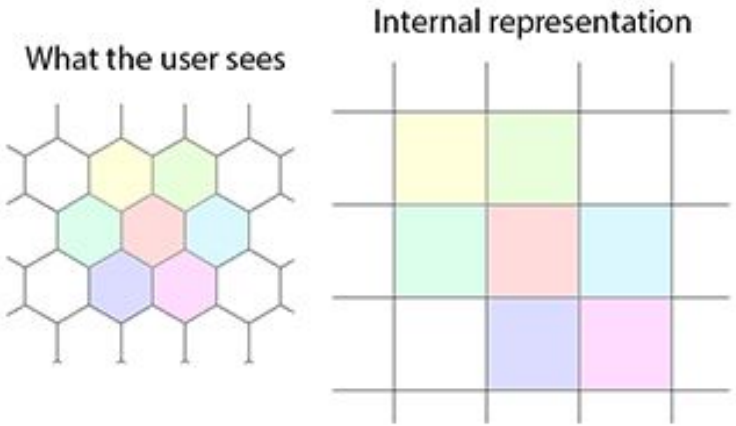
```
public static void main(String[] abc)  {
    Scanner sc = new Scanner(System.in);
    String firstline = sc.nextLine();
    StringTokenizer st = new StringTokenizer(firstline);

    h = Integer.parseInt(st.nextToken());
    n = Integer.parseInt(st.nextToken());
    m = Integer.parseInt(st.nextToken());

    hive = new String[n][m+1];
    answer = new int[n*(m+1)];

    for(int i=0;i<n;i++) {
        String line = sc.nextLine();
        //Split with spaces
        String parts[] = line.split(" ");

        if(i%2==0) {
            // End with a space if odd
            hive[i][m]="";
            for(int j=0;j<parts.length;j++) {
                hive[i][j]=parts[j];
            }
        }
        else {
            // Start with a space if even
            for(int j=0;j<parts.length;j++) {
                hive[i][j]=parts[j];
            }
        }
    }
```



Main method

```
for(int i=0;i<n;i++) {
    for(int j=0;j<m+1;j++) {
        if(hive[i][j].equals(".")) {
            number=0;
            DFS(i,j);
            answer[counter] = number;
            counter++;
        }
    }
}
```

DFS

```
public static void DFS(int row,int col) {
    hive[row][col]="#";
    number++;

    if(row%2==0) {
        //Odd case
        check(row+1,col);
        check(row+1,col+1);
        check(row-1,col);
        check(row-1,col+1);
        check(row,col-1);
        check(row,col+1);
    }
    else {
        // Even case
        check(row-1,col);
        check(row-1,col-1);
        check(row+1,col);
        check(row+1,col-1);
        check(row,col-1);
        check(row,col+1);
    }
}
```

Other Ideas

- BFS
- Converting the array to a graph of nodes

Test Cases

- Invalid Testing
- Boundary Testing
- Valid Testing

Examples:

- We checked that the output would be 0 for inputs with every cell being filled.
- We checked that the edges of the odd and even rows were being checked using print statements.
- We checked if the output is 0 if the hive size was less than the amount of honey.
- We checked that the output is 1 if the hize size was more than the amount of honey, and every cell was empty.

Section 3: Kattis Performance