Editorial

quangvn2508

March 18, 2020

In this editorial, I would like to provide some approachs for each questions. These solution are not the only or the best and most optimised solution. If you have a better solution and want to share please feel free to message our **facebook group** and we will update the solution to other contestants.

1 Question 1: Smallest String (Easy) [problem link]

Setter: quangvn2508
Solution by kanishkalikhanna

- Time complexity: O(nm)

Go through each string and calculate the ASCII value. Note that the length of string might not always determine the smallest string.

Solution (Python 3):

```
n = int(input())
data = [str(i) for i in input().split()]

smallest = ""
score = float('inf')
for s in data:
    string_score = 0
    for i in s:
        string_score += ord(i)
    if string_score < score:
        score = string_score
        smallest = s

print(smallest)
```

2 Question 2: Shortest path in maze (Medium) [problem link]

Setter: quangvn2508
Solution by quangvn2508

- Time complexity: O(n^2)
- Flood fill, BFS, DP

Mark all cells in maze as unvisited and set the start cell to be visited. The idea is to visit all the cell in group distance 1 from the start (i.e any available cells that next to the start position) and mark all of them visited. Again, visit all the cells that in group distance 2, 3, etc and mark them visited. When the end position is marked visited, then the current group denote the shortest length of the path between start and end position.

Let define some symbol used in the algorithm:

- *Queue* is a queue of coordinate (x,y)
- $visited[x][y]$ is a boolean indicate whether the cell at coordinate (x, y) is visited.
- $Start[2] = [x, y]$ is an array with two elements denoted the x and y coordinate of the start position.
- $End[2] = [x, y]$ is an array with two elements denoted the x and y coordinate of the end position.

Algorithm

$directions \leftarrow [[0,1],[0,-1],[1,0],[-1,0]]$

$visited[Start[0]][Start[1]] = True$

$distance\_from\_start \leftarrow 0$

PUSH *Start* to *Queue*

WHILE NOT $visited[End[0]][End[1]]$:

    $distance\_from\_start \leftarrow distance\_from\_start + 1$
    $s \leftarrow sizeofQueue$
    FOR i = 1, 2, ..., s:
        $current_cell \leftarrow popfirstelementinQueue$
        FOR each direction in *directions*:
            $new\_x \leftarrow current_cell[0] + direction[0]$
            $new\_y \leftarrow current_cell[1] + direction[1]$
            IF ($position(new\_x, new\_y)$ is $'0'$) AND ($visited[new\_x][new\_y]$ is $False$):
                $visited[new\_x][new\_y] = True$
                PUSH (new_x, new_y) to *Queue*

RETURN $distance\_from\_start$

Each while loop will visit all cells that have the same distance from start node. Therefore, distance_from_start is increased for each repetition of while loop.

Solution (C++)

```
#include <iostream>
#include <queue>
#include <cstring>

using namespace std;

const int direction[4][2] = {{0,1}, {-1,0}, {0, -1},{1,0}};

size_t sx, sy;

bool maze[6000][6000];
bool visited[6000][6000];
/*
    -2 = wall
```

```
    -1 = path
*/


void pathfinding(int x, int y, int ex, int ey)
{
    pair<int, int> s, e;
    s = {x,y};
    e = {ex, ey};

    queue<pair<int, int>> q;

    memset(visited, false, sizeof(visited));
    visited[s.first][s.second]=true;
    unsigned int depth = 0;
    q.push(s);
    while (!visited[e.first][e.second]) {
        depth++;
        int q_size = q.size();
        for (int i = 0; i < q_size; i++) {
            pair<int, int> cur = q.front(); q.pop();
            for (int d = 0; d < 4; d++) {
                int newx = cur.first + direction[d][0];
                int newy = cur.second + direction[d][1];

                if (!(newx > 0 && newx < sx - 1 && newy > 0 && newy < sy - 1)) continue;
                if (!maze[newx][newy] && !visited[newx][newy]) {
                    visited[newx][newy] = true;
                    q.push({newx, newy});
                }
            }
        }
    }
    cout << depth << endl;
}

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> sx >> sy;

    int x,y,ex,ey;
    cin >> x >> y >> ex >> ey;
    for (int i = 0; i < sx; i++) {
        string s; cin >> s;
        for (int j = 0; j < sy; j++) maze[i][j] = (s[j] == '1');
    }
    pathfinding(x,y,ex,ey);

    return 0;
}
```

3 Question 3: Tower of Hanoi (Hard) [problem link]

Setter: quangvn2508
Solution by moxis [link]

- Time complexity: O(2^n)
- Recursion, DFS

Break the problem into smaller problems. Moving n disks from rod 1 to rod 3 can be broke down into 3 steps (move n-1 disk from rod 1 to rod 2, move disk nth from rod 1 to rod 3, move n - 1 disks from rod 2 to 3). By applying this recursively, the problem become easier.

Let generalise the problem furthermore. Give 3 rods (start_rod, destination_rod, and ,mid_rod). The task is to move all k disks, from start_rod to destination_rod. In order to achieve this, we define the function F(k, start_rod, end_rod) which execute the following steps.

- step 1: we move k - 1 disks from start_rod to mid_rod (deeper function F(k - 1, start_rod, mid_rod))
- step 2: the kth disks from start_rod to destination_rod
- step 3: we move k - 1 disks from mid_rod to destination_rod (deeper function F(k - 1, mid_rod, destination_rod))

Notice that step 1 and 3 are deeper recursive call where mid_rod become the next destination_rod (step 1) and the next start_rod (step 3). However, when moving n - 1 disk to rod 2, we need to move n - 2 disk to rod 3 first. So how can we determine when to move to which rod?

Since we have only 3 rod, let list all the possibilities.

- 1, 2 $\rightarrow$ 0
- 0, 2 $\rightarrow$ 1
- 0 ,1 $\rightarrow$ 2

We can obtain this using the following equation

$$mid\_rod = 3 - (start\_rod + destination\_rod)$$

Finally, we define the base case for our recursive function. When $k = 1$ we just need to move disk 1 from start to destination rod.

Algorithm

1 **function F(k, start_rod, destination_rod)**
2 IF $k = 1$:
3 MOVE disk 1 from start_rod to destination_rod
4 PRINT current state
5 ELSE:
6 mid_rod $\leftarrow$ 3 - (start_rod + destination_rod)
7 F(k - 1, start_rod, mid_rod)
8 MOVE disk kth from start_rod to destination_rod
9 PRINT current state
10 F(k - 1, mid_rod, destination_rod)

We need to print the state of the game every time we move a disk (line 4, 9)

Solution (Python 3)

```
state = [[],[],[]]

def print_state():
    for i in state:
        if i:
            print(" ".join(i))
        else:
```

```
            print()

def toh(n, a, b, c):
    if n == 1:
        state[b].append(state[a].pop())
        print_state()
        return



    toh(n-1, a, c, b)
    state[b].append(state[a].pop())
    print_state()
    toh(n-1, c, b, a)

N = int(input())
state[0] = list(reversed([str(i) for i in range(1, N + 1)]))
print_state()

toh(N, 0, 2, 1)
```
