CyberChallenge.IT 2019 - Programming
Commented solutions

Contents

1 Problem "Unlock"

1.1 Problem Statement

A cybersecurity firm has to develop an access control device to unlock a door. Unlocking requires a magnetic square card to be placed at an appropriate position over a square pad next to the door.
The problem is modeled as follows:

- The **card** is a $n \times n$ 0/1 matrix, with $n \geq 1$
- The **pad** is a $m \times m$ 0/1 matrix, with $m \geq 1$
- The card unlocks the door if it aligns with the pad, i.e., it appears as a submatrix of the pad up to rotations.

A $m \times m$ matrix cell has coordinate (i, j) if it lies on row i and column j, for $0 \leq i < m$ and $0 \leq j < m$.
Example:

A 3×3 card:

0 1 0
0 1 1
0 1 0

A 5×5 pad:

0 0 0 0 0
0 0 0 1 0
0 0 1 1 1
0 0 0 0 0
0 0 0 0 0

The card unlocks the door if it is rotated by 270 deg clockwise and its upper-left corner is placed at coordinates $(1, 2)$ of the pad:

0 0 0 0 0
0 0 **0 1 0** ← Card aligns here after 270 deg clockwise rotation
0 0 **1 1 1**
0 0 **0 0 0**
0 0 0 0 0

Write a program that checks whether a card aligns with a pad. In case there are more possibilities of unlock the pad you must return the one with the lowest coordinates and the lowest rotation.

1.2 Problem Details

Input

Your program must read the input data from the standard input.
The first two lines of the input contain **n** and **m**, respectively.
Then, **n** lines follow, representing the rows of the card, each with **n** `'0'/'1'` consecutive chars.
Finally, **m** lines follow, representing the rows of the card, each with **m** `'0'/'1'` consecutive chars.

Output

Your program must write the output data into the standard output.
The output must contains the three integer **i j r** if the card unlocks the pad if placed in (i, j) rotated by r deg, the string **err** if the card does not align with the pad.

Scoring

For each of the test cases the program will be tested for, the following constraints are met:

- $1 \leq n \leq 16$.
- $1 \leq m \leq 16$.

1.3 Solution

This is an implementation task. No clever idea is necessary to solve it, but it requires a good familiarity with a programming language.
The solution of this problem consist in checking, for all the possible positions and rotations, if the card matches the pad.

1.4 Source Code

C++

```cpp
#include <iostream>
using namespace std;

int main(){
    int n, m;
    cin >> n >> m;
    if (n > m){
        cout << "err\n";
    return 0;
    }

    char card[n][n+1], pad[m][m+1];
    for (int i = 0; i < n; i++){
            cin >> card[i];
    }
    for (int i = 0; i < m; i++){
                cin >> pad[i];
    }

    for (int i = 0; i + n - 1< m; i++){
        for (int j = 0; j + n - 1< m; ++j){
            int k=0, l=0;
            bool found=1;
            for (k = 0; k < n; ++k){
                for (l = 0; l < n && found; ++l){
                    if(pad[k + i][l + j] != card[k][l])
                        found = 0;
                }
            }
            if(found){
                cout << i << ' ' << j << ' ' << 0 << endl;
                return 0;
            }

            found = 1;
            for (l = 0; l < n; ++l){
                for (k = n - 1; k >= 0 && found; --k){
                    if(pad[l + i][n - 1 - k + j] != card[k][l])
                        found = 0;
                }
            }
            if(found){
                cout << i << ' ' << j << ' ' << 90 << endl;
                return 0;
            }

            found = 1;
            for (k = n - 1; k >= 0 ; --k){
                for (l = n - 1; l >= 0 && found; --l){
                    if(pad[n - 1 - k + i][n - 1 - l + j] != card[k][l])
                        found = 0;
                }
            }
            if(found){
                cout << i << ' ' << j << ' ' << 180 << endl;
                return 0;
            }
```

```
58
59              found = 1;
60              for (l = n - 1; l >= 0; --l){
61                  for (k = 0; k < n && found; ++k){
62                      if(pad[n - 1 - l + i][k + j] != card[k][l])
63                          found = 0;
64                  }
65              }
66              if(found){
67                  cout << i << ' ' << j << ' ' << 270 << endl;
68                  return 0;
69              }
70          }
71      }
72      cout << "err\n";
73      return 0;
74  }
```

Python

```
1   #!/bin/env python3
2
3   import sys
4
5   n, m = map(int, input().split(" "))
6
7   card = []
8
9   for i in range(n):
10      r = input()
11      card.append([j for j in r])
12
13  pad = []
14
15  for i in range(m):
16      r = input()
17      pad.append([j for j in r])
18
19
20  def submatrix(pad, n, m):
21      subs = dict()
22
23      steps = m - n + 1
24
25      for r in range(steps):
26          for c in range(steps):
27              sub = []
28
29              for ri in range(n):
30                  rsub = []
31                  for ci in range(n):
32                      rsub.append(pad[ri + r][ci + c])
33
34                  sub.append(rsub)
35
36              subs[str(r) + "-" + str(c)] = sub
37
38      return subs
39
```

```
40
41  def rotateCard(card, n, deg):
42      newCard = []
43
44      for i in range(n):
45          newCard.append([0 for i in range(n)])
46
47      if deg == 0:
48          return card
49      if deg == 90:
50          for i in range(n):
51              for j in range(n):
52                  newCard[i][j] = card[n - j - 1][i]
53      if deg == 180:
54          for i in range(n):
55              for j in range(n):
56                  newCard[i][j] = card[n - i - 1][n - j - 1]
57      if deg == 270:
58          for i in range(n):
59              for j in range(n):
60                  newCard[i][j] = card[j][n - i - 1]
61
62      return newCard
63
64
65  def match(c1, c2, n):
66      for i in range(n):
67          for j in range(n):
68              if c1[i][j] != c2[i][j]:
69                  return False
70
71      return True
72
73
74  def testCard(subs, card, n):
75      for coord, sub in subs.items():
76          for r in range(4):
77              if match(rotateCard(card, n, r * 90), sub, n):
78                  return coord, r * 90
79
80
81  res = testCard(submatrix(pad, n, m), card, n)
82
83  if res is None:
84      print("err")
85  else:
86      print(res[0].split('-')[0], res[0].split('-')[1], res[1])
```

2 Problem "Malware"

2.1 Problem Statement

A malware creator needs to be able to control where a certain number of samples of the particularly harmful malware strain *av23* are located on a $n \times n$ grid-connect network of computing devices. To this aim, they need to be able to move each of them individually along the grid.
Before implementing the actual infrastructure, the malware creator needs to run some tests on a simulated grid network to check the effects of his evil plan.
One of the problems they have to simulate can be modeled as follows:

1. the grid is represented as $n \times n$ board that contains up to 10 malware samples identified by characters between 0 and 9. Empty locations are marked with a non-digit character.
2. Each sample can be moved a certain number of times in any of the four directions: up (U), down (D), left (L) and right (R). No diagonal moves are allowed.
3. A sample can only be moved towards an empty location and cannot escape the board's boundaries.
4. When a malware sample is moved, its previous location becomes empty.

You have to stop the malware creator. To do so, you have to be able to reproduce yourself what they would do to run the simulation.
More specifically, you are given in input the size n of the board, the $n \times n$ matrix of chars, the number m of move operations and the list of m move operation. Each move operation is defined by: SAMPLE_ID (id of the sample to be moved), DIR (movement direction as U, D, L or R) and REP (number of repeated moves to be performed).
After each move operation you have to print the return code of the operation as follow:

- 0: malware sample successfully moved.
- 1: error: invalid number of repetitions (REP < 1).
- 2: error: SAMPLE_ID does not appear on the board.
- 3: error: attempt to move the sample over a non-empty location.
- 4: error: attempt to move the sample past the board's boundaries.

In case of multiple errors return the one with the lower code.
At the end of the m operations you have to print the final board.

Example:
Given a board 5×5 containing 3 samples 1, 7 and 9:

```
. . . . .
. 1 . . .
. . . 9 .
. . . . .
. . . 7 .
```

Execute the following operations:

7 U 3 (move the sample 7 up of 3 positions)
Return code 3 (moving up sample 7 eventually hits 9)

9 L 3 (move the sample 9 left of 3 positions)
Return code 0 as it is possible to move 9 left of 3 positions, the final board is thus:

```
. . . . .
. 1 . . .
9 . . . .
. . . . .
. . . 7 .
```

2.2 Problem Details

Input

Your program must read the input data from the standard input.
The first line of the input contains the integers **n** and **m**, space separated, the size of the board and the number of moves operations.
Then, **n** lines follow, representing the rows of the board, each with **n** characters.
Finally, **m** lines follow, representing SAMPLE_ID DIR REP, an integer, a character and an integer, space-separated describing a move operation.

Output

Your program must write the output data into the standard output.
The output must contains **m** lines containing return code of the **m** move operations, one for line, followed by **n** lines containing a string of n characters each, representing a single row of the final board.

Scoring

For each of the test cases the program will be tested, the following constraints are met:

- **Subtask 1** [**20 points**]: $n, m \leq 10$.
- **Subtask 2** [**30 points**]: $n, m \leq 100$.
- **Subtask 3** [**50 points**]: $n, m \leq 1.000$.
- $1 \leq n \leq 1.000$
- $1 \leq m \leq 1.000$
- DIR $\in \{U, D, L, R\}$

2.3 Solution

This problem is slightly more difficult than the second one, since a little optimization is required. The first subcases does not require particularly complex ideas, while the last one requires a more optimized solution to be thought of. Data structures more complex than stacks should not be necessary.

Subcase 1 and 2 (20+30 points)

The simplest idea to solve this problem is to store the grid in memory. Then, for each operation, the position of the sample is retrieved iterating on the grid. The operation is then applied, checking for errors and updating the grid. The complexity of this approach is $O(n^2 m)$

Subcase 3 (50 points)

The previous solution can be optimized removing the iteration over the grid to search the sample. It is sufficient to store into an array the positions of the samples and access them in constant time. The complexity is thus reduced to $O(m)$.

2.4 Source Code

C++

```cpp
#include <iostream>
using namespace std;

int main(){
    int m,n;
    int s,r;
    char d;
    int x, y, x2, y2;
    char grid[1005][1005];
```

```
    int pos[10][2] = {{-1,-1},{-1,-1},{-1,-1},{-1,-1},{-1,-1},{-1,-1},{-1,-1},{-1,-1},{-1,-1},{-1,-1}};

    cin >> n >> m;

    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            cin >> grid[i][j];
            if(grid[i][j] != '.'){
                pos[grid[i][j]-'0'][0] = i;
                pos[grid[i][j]-'0'][1] = j;
            }
        }
    }

    for(int i=0; i<m; i++){
        cin >> s >> d >> r;

        if(r < 1){
            cout << "1\n";
            continue;
        }

        x2 = x = pos[s][1];
        y2 = y = pos[s][0];

        if(x == -1 && y == -1){
            cout << "2\n";
            continue;
        }

        bool non_empty = false, out_bound = false;

        for(int j=0; j<r; j++){
            switch(d){
                case 'L':
                    x--;
                    break;
                case 'R':
                    x++;
                    break;
                case 'U':
                    y--;
                    break;
                case 'D':
                    y++;
                    break;
            }

            if(y<0 || x<0 || y>=n || x>=n){
                out_bound = true;
                break;
            }
            if(grid[y][x] != '.'){
                non_empty = true;
                break;
            }
        }

        if(non_empty){
            cout << "3\n";
            continue;
        }
```

```
        if(out_bound){
            cout << "4\n";
            continue;
        }

        cout << "0\n";
        grid[y][x] = grid[y2][x2];
        grid[y2][x2] = '.';
        pos[s][0] = y;
        pos[s][1] = x;
    }

    for(int i=0; i<n; i++){
        cout << grid[i] << "\n";
    }

    return 0;
}
```

Python

```
#!/bin/env python3

n, m = map(int, input().split(" "))

grid = []

pos = [[-1, -1] for _ in range(10)]

for y in range(n):
  grid.append(list(input()))

  for x in range(n):
    if grid[y][x] == '.':
      continue
    pos[int(grid[y][x])] = [y, x]

def find(s):
  s = int(s)
  return pos[s][0], pos[s][1]

for _ in range(m):
  s, d, r = input().split(" ")
  r = int(r)

  if r < 1:
    print(1)
    continue

  y, x = find(s)
  y2, x2 = y, x

  if x == -1 and y == -1:
    print(2)
    continue

  non_empty = False
  out_bound = False

```

```
  for __ in range(r):
    if d == 'L':
      x = x-1
    elif d == 'R':
      x = x+1
    elif d == 'U':
      y = y-1
    elif d == 'D':
      y = y+1

    if y < 0 or x < 0 or y >= n or x >= n:
      out_bound = True
      break
    if grid[y][x] != '.':
      non_empty = True
      break

  if non_empty == True:
    print(3)
    continue

  if out_bound == True:
    print(4)
    continue

  print(0)
  grid[y][x] = grid[y2][x2]
  grid[y2][x2] = '.'
  pos[int(grid[y][x])] = [y, x]

for g in grid:
  print("".join(g))
```

3 Problem "Reverse"

3.1 Problem Statement

Alice and Bob wish to make fun of their friends by letting they think they can fluently write to each other in some weird foreign language.
They want to use an algorithm A that can be applied both to scramble and to recover the original message, that is, for any given text t, $A(A(t)) = t$.
The algorithm should confuse the reader by leaving a certain amount of semi-intelligible meaning.
On a sunny terrace in Amsterdam, sipping a cup of tea, they think of a simple solution that works for any text on an alphabet of a given size, say 256 (well yeah, good ol' 8-bit ASCII).
Here's the idea. They take the text t and compute the frequency $f(c)$ of each character c of the alphabet in t. For each distinct frequency value, they consider all characters having exactly that frequency and they sort them by ASCII code. In each resulting group, they exchange the ASCII codes of the smallest with the largest, the second smallest with the second largest, etc.

Example:
Given the text $t =$ "EAEEABDBFBCDF", algorithm A obtains the frequency groups:

- $f('A') = f('D') = f('F') = 2$
- $f('B') = f('E') = 3$
- $f('C') = 1$
- All other frequencies are 0

Now, the transformation by reverse ASCII code will replace characters as follows:

- $A \to F$, $B \to E$, $C \to C$
- $D \to D$, $E \to B$
- $F \to A$,

Hence, $t' = A(t) =$ "BFBBFEDEAECDA".
Since the transformation preserves the frequency of characters, the same algorithm A applied back to t' yields $t = A(t') =$ "EAEEABDBFBCDF". Neat huh?

Write a program that, given an ASCII text t as input, produces the ASCII text $t' = A(t)$ as output.

3.2 Problem Details

Input

Your program must read the input data from the standard input.
The first line of the input contains the integer **n**, representing the number of characters in the message **t**
The second line contains **n** ASCII characters, representing the message **t**.

Output

Your program must write the output data to the standard output.
The output should consists of only one line, containing the decrypted message **t'**.

Scoring

For each of the test cases the program will be tested for, the following constraints are met:

- $1 \leq n \leq 100.000$.
- All the characters of the message t are printable.

3.3 Solution

The solution to this problem goes through the following steps:

- compute the frequencies of the characters in the message
- while doing so, group the characters with the same frequency
- sort the groups in descending order, based on their ASCII code
- apply the substitution

3.4 Source Code

C++

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int n;
    string s;
    char m[256];
    vector<char> rm[100000];

    cin >> n;
    cin >> s;


    set<char> scs(s.begin(), s.end());
    string cs;
    for(auto c: scs){
        cs += c;
    }
    sort(cs.begin(), cs.end());

    for(auto c: cs){
        int f = 0;
        for(auto x:s){
            if(x==c){
                f++;
            }
        }
        rm[f].push_back(c);
    }

    for(auto fr: rm){
        vector<char> seq(fr.size()), seqr(fr.size());
        for(int i=0; i<fr.size(); i++){
            seq[i] = fr[i];
            seqr[fr.size()-i-1] += fr[i];
        }
        for(int i=0; i<seq.size(); i++){
            m[seq[i]] = seqr[i];
        }
    }

    for(auto c: s){
        cout << char(m[c]);
    }
    cout << endl;
    return 0;
}
```

Python

```python
#!/bin/env python3


n = int(input())
s = input()

cs = list(sorted(set(s)))
m = {}
rm = {}

for c in cs:
  f = sum([1 if x == c else 0 for x in s])
  if f not in rm:
    rm[f] = []
  rm[f].append(c)

for fr in rm:
  seq = "".join(rm[fr])
  seqr = seq[::-1]
  for i in range(len(seq)):
    m[seq[i]] = seqr[i]

print("".join(m[x] for x in s))
```