The Grid Search

locked

by PRASHANTB1984

Problem | Submissions | Leaderboard | Discussions | Editorial

Editorial by PRASHANTB1984

There are 2 approaches demonstrated here.

One is the C++ Program which does an element by element match for the smaller array in the larger 2D and breaks as soon as there is a mismatch.

The second one is a short but tricky Ruby script which uses a regular expression based approach to search for the smaller pattern inside the larger one. It will be an interesting exercise for the reader to brainstorm on how and why it works!

Featured Solutions

C++

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

int M, N, R, C;
vector <string> g, pat;
bool present(int x, int y) {
    for(int i = 0; i < (int)R; ++i) {
        for(int j = 0; j < (int)C; ++j) {
            int ii = i + x, jj = j + y;
            if(ii >= M || jj >= N)
                return false;
            else if(g[ii][jj] != pat[i][j])
                return false;
            else
                continue;
        }
    }
    return true;
}

int main()
{
int t;
cin >> t;
assert(t >= 1 && t <= 5);

while(t--) {
    cin >> M >> N;
    g = vector <string> (M);
    for(int i = 0; i < (int)M; ++i) {
        cin >> g[i];
    }

    cin >> R >> C;
    pat = vector <string>(R);
    for(int i = 0; i < (int)R; ++i) {
        cin >> pat[i];
    }

    bool isPresent = false;

    for(int i = 0; i < (int)M; ++i) {
        for(int j = 0; j < (int)N; ++j) {
            isPresent = present(i, j);
            if(isPresent)
                break;
        }
        if(isPresent)
```

Statistics

Difficulty: **Medium**

Publish Date: **Oct 01 201**

```
}
```

Python 2

```
# iterate over every element in grid
for _ in range(input()):
    big = []
    small = []
    R, C = map(int, raw_input().split())
    for __ in range(R):
        big.append(raw_input())
    r, c = map(int, raw_input().split())
    for __ in range(r):
        small.append(raw_input())
    found = False
    for i in range(R - r + 1):
        for j in range(C - c + 1):
            flag = True
            for k in range(r):
                for l in range(c):
                    if big[i + k][j + l] != small[k][l]:
                        flag = False
                        break
                if not flag:
                    break
            if flag:
                print "YES"
                found = True
                break
        if found:
            break
    else:
        print "NO"
```

Python 2

```
# search based solution
for _ in range(input()):
    big = []
    small = []
    R, C = map(int, raw_input().split())
    for __ in range(R):
        big.append(raw_input())
    big_str = "".join(big)
    r, c = map(int, raw_input().split())
    for __ in range(r):
        small.append(raw_input())
    small_str = small[0]
    # get all starting positions of first line of small grid
    t = [i for i in range(len(big_str)) if big_str.startswith(small_str, i)]
    for i in t:
        flag = 1
        row = i / C
        col = i % C
        if row > R - r:
            flag = 0
            continue
        for j in range(1,r):
            if small[j] != big[row+j][col: col + c]:
                flag = 0
                break
        if flag == 1:
            print "YES"
            break
    else:
        print "NO"
```

Ruby

```
# regex based solution
N = gets().to_i
(1..N).each do |num|
```

```ruby
  # Read in the smaller 2D grid
  r2, c2 = gets().split().map{|x|x.to_i}
  regular = ""
  (1..r2).each do |row|
    line = gets()
    regular += line.strip + "[0-9]{#{c - c2}}"
  end

  # Create a regular expression from the search pattern

  r = Regexp.new regular

  position = text.index(r)

  # We don't want to match a case where this pattern might match a
  # block which crosses the edges of the larger 2D grid - we want to
  # match the pattern strictly inside the larger 2D grid

  while (position != nil) && ((position % c + c2) > c)
    warn "looping"
    position = text.index(r, position + 1)
  end
  if position == nil
    puts "NO"
  else
    row = (position / c) + 1
    col = (position % c) + 1
    puts "YES"
  end
end
```