Design and Analysis of Algorithms Assignment - 5

Department of Information Technology,

Indian Institute of Information Technology, Allahabad 211015, India

SUBHASH BALLA(IIT2019207), DHANUSH VASA (IIT2019208)

INTRODUCTION

Given a matrix a size n times m,the task is to find maximum length snake sequence and print any of the sequence with maximum length.Also there are some conditions that need to be obeyed for a sequence to be a snake sequence. This problem is to be done with dynamic programming paradigm.

Dynamic Programming - This algorithmic technique can be used to solve problems of types-

1)optimisation problems(our problem is of this type)
2)decision problems
3)counting no.of ways

We can think of it as a careful brute force approach.we will iterate over all possible solutions and choose the best solution similar to naive algorithm,but once we find a solution to a sub problem(of course of similar type as actual problem),we store it in memory so that whenever we required that solution,need not compute it again.

This can greatly influence the run time of our algorithm.A lot of algorithmic problems which runs in exponential time when implemented naively can be run in polynomial time with dynamic programming paradigm

ALGORITHM DESIGN

Given a matrix a[][] of size n*m.
Define dp[][] of size n*m where,

dp[i][j] stores maximum snake sequence length starting at (i,j).

From the question,it is clear that a snake starting at a a position (say (i,j)) can only move right or down.

We will traverse dp array in such a a way that while we are at position (i,j),we would have already calculated dp(i,j+1) and dp(i+1,j).It's because that dp(i,j) requires dp(i+1,j) and dp(i,j+1).
One possibility is to traverse from bottom-right to top-left.

conditions -
we can move from

$$(i,j)->(i,j+1) <=> a[i][j]-a[i][j+1] = (+|-)1$$
$$(i,j)->(i+1,j) <=> a[i][j]-a[i+1][j] = (+|-)1$$

Possibiltites -

1) (!right possible AND !down possible), dp[i][j]= 1 (snake starts and ends at (i,j))
2) (right possible AND down possible), dp[i][j]= 1+max(dp[i][j+1],dp[i+1][j])
3) (right possible AND !down possible), dp[i][j]=1+dp[i][j+1]
4) (!right possible AND down possible), dp[i][j]=1+dp[i+1][j]

Base cases -

dp[n][m]=1 (no more cells right or down of (n,m)).
right most column - snake can only go down.
bottom most row - snake can only go right.

Final Answer -

max{dp[i][j]} $\forall\ 1 <= i <= n$, $1 <= j <= m$

Trace path -

To trace the path of maximum snake sequence,
we maintain a matrix **path[][]** where **path[i][j]** points to either **RIGHT** or **DOWN** cell(which ever of dp(i+1,j) ,dp(i,j+1) yields maximum) or **NONE** ,if it has to end there it self.

Code Implementation

```cpp
#include<bits/stdc++.h>
using namespace std;
#define IOS ios_base::sync_with_stdio(false),cin.tie(NULL)
#define mod 1000000007
#define  pb push_back
typedef long long ll;
typedef unsigned long long ull;
typedef  long double ld;
typedef vector<int> vi;
typedef vector<bool> vb;
typedef vector<vi >vvi;
typedef vector<ll> vll;
typedef vector<vll >vvll;


ll n, m;
ll maxN = 101, maxnumber = 5;
ll pos_i = 1, pos_j = 1, max_length = 1;
ll a[1001][1001], dp[1001][1001];
ll path[1001][1001], path_table[1001][1001];


void print_path() {


        while (true) {
                path_table[pos_i][pos_j] = a[pos_i][pos_j];
                if (path[pos_i][pos_j] == 1)pos_j++;
                else if (path[pos_i][pos_j] == -1)pos_i++;
                else break;
        }

        for (ll i = 1; i <= n; i++) {
                for (ll j = 1; j <= m; j++)cout << path_table[i][j] << " ";
                cout << "\n";
        }
}


void solve() {


//base_cases--->

        dp[n][m] = 1;
        path[n][m] = 0;


//bottom most row
        for (ll i = m - 1; i >= 1; i--) {
                if (a[n][i] == a[n][i + 1] + 1 || a[n][i] == a[n][i + 1] - 1) {
                        dp[n][i] = dp[n][i + 1] + 1, path[n][i] = 1;
                        if (dp[n][i] > max_length)max_length = dp[n][i], pos_i = n, pos_j = i;
                }
                else dp[n][i] = 1;
        }


//right most coulmn
        for (ll i = n - 1; i >= 1; i--) {
                if (a[i][m] == a[i + 1][m] + 1 || a[i][m] == a[i + 1][m] - 1) {
                        dp[i][m] = dp[i + 1][m] + 1, path[i][m] = -1;
                        if (dp[i][m] > max_length)max_length = dp[i][m], pos_i = i, pos_j = m;
                }
                else dp[i][m] = 1;
        }



for (ll i = n - 1; i >= 1; i--) {
        for (ll j = m - 1; j >= 1; j--) {

  bool right_possible = false, down_possible = false;

 if (a[i][j] == a[i][j + 1] + 1 || a[i][j] == a[i][j + 1] - 1)right_possible = true;
 if (a[i][j] == a[i + 1][j] + 1 || a[i][j] == a[i + 1][j] - 1)down_possible = true;

if (!right_possible && !down_possible) {
                        dp[i][j] = 1;
                        continue;
 }

if (right_possible)dp[i][j] = dp[i][j + 1] + 1, path[i][j] = 1;

if (down_possible && dp[i + 1][j] + 1 > dp[i][j])dp[i][j] = dp[i + 1][j] + 1, path[i][j] = -1;

if (dp[i][j] > max_length)max_length = dp[i][j], pos_i = i, pos_j = j;
                }
        }

cout << "maximum length is ---> " << max_length << "\n";
cout << "path--->\n";
print_path();
}

int main() {

IOS;
srand(time(0));

n = rand() % maxN + 1, m = rand() % maxN + 1;

cout << "matrix width and height are--> \n" << n << " " << m << "\n";

for (ll i = 1; i <= n; i++)for (ll j = 1; j <= m; j++)a[i][j] = rand() % maxnumber + 1;

cout << "\ninput matrix is ---> \n";
        for (ll i = 1; i <= n; i++) {
                for (ll j = 1; j <= m; j++)cout << a[i][j] << " ";
                cout << "\n";
        }

solve();
}
```

ALGORITHM ANALYSIS

Number of sub problems=**n*m.**
Time per subproblem=**O(1)** (only finding maximum of 2 values).
Hence **T(n,m)=(n*m)*O(1).**

=O(n*m)

We have created **dp**[][] and **path**[][] of n*m size.
Hence **auxillary space=O(n*m)**

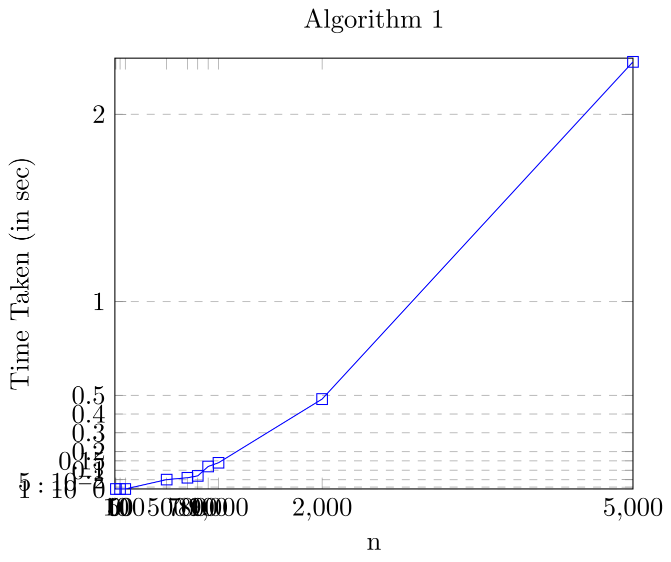
However,If we don't want to trace the path and our only objective is to find maximum snake sequence length,then we can optimise space to **O(m)**.Its because that that only (i+1)th row is sufficient to clalculate ith row.

---

1 Experimental Analysis

In the following table some cases are plotted for the first algorithm,

m	n	Time Taken (in ms)
10	10	0.0000
50	50	0.0000
100	100	0.0000
500	500	0.0500
700	700	0.0600
800	800	0.0700
900	900	0.1200
1000	1000	0.1400
2000	2000	0.4800
5000	5000	2.2800



CONCLUSION

We can conclude that the given problem of finding maximum length snake sequence and tracing the path is acheieving the time and space complexities of **O(n*m)**.

ACKNOWLEDGMENT

We are very much grateful to our Course instructor Mr.Rahul Kala and our mentor, Ms.Tejasvee, who have provided the great opportunity to do this wonderful work on the subject of Design and Analysis of Algorithms, specifically on the algorithmic paradigm of Dynamic Programming.

NOTE

1)One can adjust the maximum values of n and m by changing **maxN** and matrix elements with **maxnumber** variables which are declared as global variables.
2)The submitted code contains comments.One can check what's happening at each step.

REFERENCES

1. Introduction to Algorithms by Cormen,Charles, Rivest and Stein.
   https://web.ist.utl.pt/ fabio.ferreira/material/asa