CS 401: Midterm EC

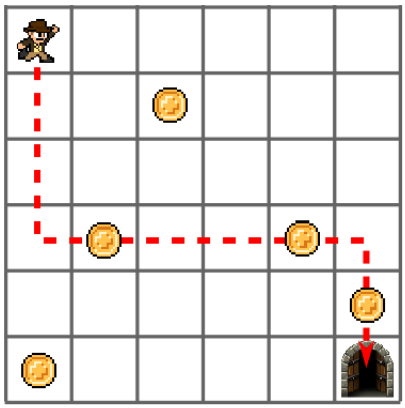
Due on April 1, 2024 at 11:59pm

Professor Sidiropoulos 9:30am

Ryan Magdaleno
rmagd2@uic.edu

Problem 1: Indiana Jones and the temple of doom.

Let A be an integer array with n columns and n rows. The array encodes the map of a room of size $n \times n$. We have $A[i,j] = 1$ if there is a gold coin at location (i,j) in the room, and otherwise $A[i,j] = 0$. Indiana Jones is initially located at position $(1,1)$, which is the north-west corner of the room, and has to reach the exit at location (n,n), which is the south-east corner. Indiana Jones is in a hurry, so he can move only south, or east. That is, at each step, starting at location (i,j) he can either move to location $(i+1,j)$, or $(i,j+1)$. Note that he cannot move diagonally.



Design an algorithm which, given the array A as input, outputs a sequence of moves for Indiana Jones, that allows him to collect the maximum possible number of gold coins before leaving the room. The running time of your algorithm should be polynomial in n.

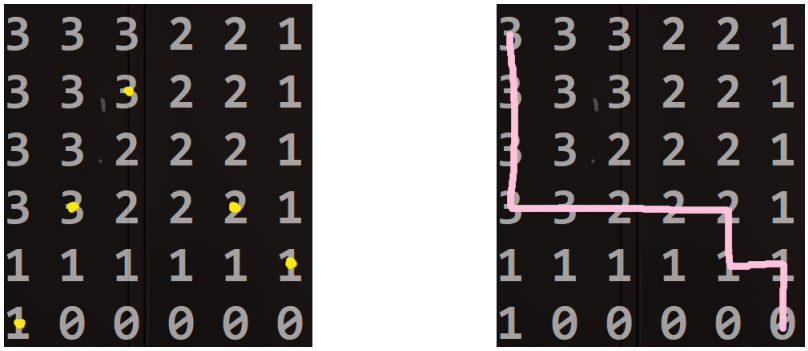
Solution
High Level Algorithm Explanation

I will employ an array called DP, it will represent the potential coin gradient values. What I mean is that for each coin, I will add one to all spaces to the left and up of this coin, including the coin's row/column. I will be using 0 indexing for this problem.

1. Create 2D DP array of size $n \times n$.
2. Reverse through A, that is start at $(n-1, n-1)$ and reverse through the last row, then go to the next row, etc, until at $(0,0)$.
3. For each DP$[i][j]$ element, take the coin value A$[i][j]$ and add it to the max of DP$[i+1][j]$ and DP$[i][j+1]$, set DP$[i][j]$ to this. Essentially we are taking the previous adjacent down and right DP values.
4. DP should now contain the gradients where the max value is at $(0,0)$ and the smallest value is at $(n-1, n-1)$.
5. Now, from DP$[0][0]$, find the path with the highest DP values. You may only move down or to the right, so we can only do DP$[i+1][j]$ or DP$[i][j+1]$. That is if DP$[i][j+1] >$ DP$[i+1]$[j] then move to the $j+1$ spot else move to the $i+1$ spot. For each movement add (j,i) to the returned path container, the indices are swapped due to the array ordering. If i or j hits their respsective last column/row, then increment the other pointer.
6. Increment the i and j pointers until i or j goes outside the range of the array, that is if i or j are ever $\geq n$.

Please remember to add index out of range checks when implementing this algorithm.

Example DP array (given scenario):



Time Complexity Justification

Let's analyze the time complexity. When setting up the DP array with the A values + gradient values, it requires checking every element, this will take $O(n^2)$ time where n is the size of one row or column.

Constructing the path requires us to move from $(0, 0)$ to $(n-1, n-1)$. Because the algorithm forces us to move either down or to the right, this will require us to move $2n$ times, therefore constructing the path will take $O(2n)$ time.

Putting every together my algorithm's time complexity is as follows:

$$T(n) = O(n^2 + 2n)$$
$$T(n) = \boxed{O(n^2)}$$

$O(n^2)$ is indeed polynomial in n, where n is the size of one row/column from A.

Correctness Justification

We make use of a DP table, where each spot represents the total number of coins from the current position to the exit, considering onlys moves to the right and down. My algorithm ensures that all positions in the grid are visited and processed in the correct order (reverse through A).

The path construction constructs the path from $(0, 0)$ to $(n-1, n-1)$. At each step we choose to move down or to the right, we move towards the element that has a higher potential coin count. This ensures that Indiana Jones will collect the max possible coins in a single path while reaching the exit and while following the movement restriction.

Real Code Implementation

```cpp
// g++ -std=c++23 -O2 -Wall EC.cpp -o EC.exe
#include <algorithm>
#include <iostream>
#include <cstdint>
#include <vector>

std::vector<std::pair<uint32_t, uint32_t>> ECProblem
    (const std::vector<std::vector<uint32_t>>& A)
{
    int32_t n = A.size();
    std::vector<std::vector<uint32_t>> DP(n, std::vector<uint32_t>(n, 0));

    // Create gradients based on each coin:
    for (int32_t i = n - 1; i >= 0; --i)
    {
        for (int32_t j = n - 1; j >= 0; --j)
        {   // Case - BR Position:
            if (i == n - 1 && j == n - 1) {
                DP[i][j] = A[i][j];
                continue;
            } // Case - I Row:
            if (i == n - 1) {
                DP[i][j] = A[i][j] + DP[i][j + 1];
                continue;
            } // Case - J Row:
            if (j == n - 1) {
                DP[i][j] = A[i][j] + DP[i + 1][j];
                continue;
            } // Take the coin value + the max adjacent axis:
            DP[i][j] = A[i][j] + std::max(DP[i + 1][j], DP[i][j + 1]);
        }
    }

    std::vector<std::pair<uint32_t, uint32_t>> path;
    int32_t i = 0, j = 0;
    // Construct path based off DP, gradient descent:
    while (i < n && j < n)
    {
        path.push_back(std::make_pair(j, i));  // Array ordering swap.
        if      (i == n-1)                    { j++; }
        else if (j == n-1)                    { i++; }
        else if (DP[i][j + 1] > DP[i + 1][j]) { j++; }
        else                                  { i++; }
    };
    return path;
}
```

```
int main()
{
    std::vector<std::vector<uint32_t>> A = {
        {0, 0, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0},
        {0, 0, 0, 0, 0, 0},
        {0, 1, 0, 0, 1, 0},
        {0, 0, 0, 0, 0, 1},
        {1, 0, 0, 0, 0, 0}
    };

    std::cout << "Maximal Path:";
    uint32_t c = 0;
    for (const auto& [i, j] : ECProblem(A))
    {
        std::cout << "\n (" << i << ", " << j << ')';
        if (A[j][i]) {
            c++;
            std::cout << " Coin.";
        }
    }
    std::cout << "\nCount: " << c;
    return 0;
}
```
