Philippe WS CSC 349

Lab 6

Problem 2

1. minCost(i, j): Represents the minimum cost required by the path to get any cell i, j;
2. minCost(i, j) = min{ minCost(i-1, j), minCost(i, j-1) }
3. It would be a m x n table, with each cell’s value contain the minimum cost to reach that cell from the starting point.
4. //Input, a m x n array of integers containing the cost required to traverse any respective cell.

findMinPath(int[m][n] matrix){

int[m][n] resultTable;

//Initialize the first two rows to allow the others to be automated.

resultTable[0][0] = matrix[0][0];

for(row = 1 to m-1)  
resultTable[row][0] = resultTable[row-1][0] + matrix[row][0];

for(col = 1 to n-1)  
resultTable[0][col] = resultTable[0][col-1] + matrix[0][col];

for(row = 1 to m-1){

for(col = 1 to n-1){

min = resultTable[row-1][col];

if(min > resultTable[row][col-1])  
min = resultTable[row][col-1];

resultTable[row][col] = min + matrix[row][col];

}

}

return resultTable;

}

1. //Input, the filled value table, cell to begin at. Assumes 0,0 is the origin

traceback(resultTable[m][n], Point (i, j)){

y = i, x = j;

Stack<Point> path;

while(y > 0 && x > 0){

min = resultTable[y-1][x];

if(min > resultTable[y][x-1])  
 min = resultTable[y][x-1];  
 path.add( Point(y, x-1) );  
 x--;

else  
 path.add( Point(y-1, x) );  
 y--;

}

if( y == 0 ){

while(x > 0)  
path.add( Point( y, x );  
x--;

}else if(x == 0){

while(y > 0)  
path.add( Point( y, x );  
y--;

}

return path;

}

1. Time complexity: , since for every row, you must fill every column.

Problem 4

1. maxRev(L) is the maximum revenue that can be obtained from a rod of length L, with segment prices pi.
2. maxRev(L) = for( i = 0 to L ): max{ maxRev(L-i) + pi }, maxRev(L<=0) = 0;
3. The table is an array of length L, with each index containing the maximum value for segments of length index+1;

//Inputs, L for length of the rod, int[] p, price per segment of length index+1 in p;  
rodMaxRevenue(int L, int[] p){

int[] result;

for i = 1 to L

for j = 1 to L

if( j <= i )  
 result[i] = max(result[i], result[i-j] + p[i]);

return result[L-1];

}

1. traceback(int[] p, int maxRev, Stack result){

for( i = p.length to 1)

if( maxRev - p[i] > 0 )

tb = traceback(p, maxRev – p[i], result)

if( tb > 0 )

result.add(p[i]);  
return 1;

else if( maxRev – p[i] < 0)  
return -1;

else  
result.add(p[i])  
return 1;

return -1;

1. Time Complexity: , for every length you try every other length.