Striver DP 11- Triangle

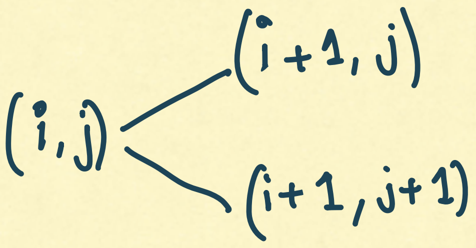
→ rows

1 (0,0)

2 (1,0)   3 (1,1)

4 (2,0)   5 (2,1)   6 (2,2)

7 (3,0)   8 (3,1)   9 (3,2)   10 (3,3)



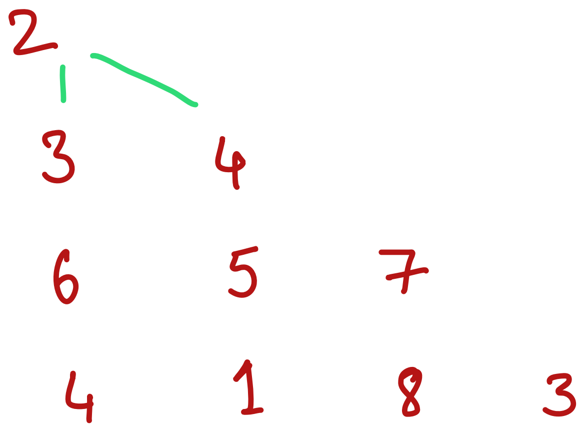
Valid Moves

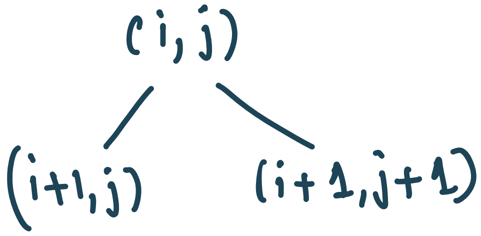
As this one has multiple end points, we might need to Write for recurrence relations which is not advisable . **so instead of that we will pick up whatever is fixed. That is the starting point** and we will write the record recurrence relation based on the starting point so that we can we need to write only one recurrence relation.

Base case - "i" should be last row

n → (n-1, n+1)

LC120- Triangle





Explore all valid paths from top to bottom, when you reach last row, pick min-cost

Recursion

```
if( i == triangle.size()-1)
    return triangle.get(i).get(j)

int down= currentVal + recursion(i+1,j)
int diag = current Val + recursion(i+1,j+1)

return min(down + diag)
```

Any recursion problem with overlapping sub-problems, can be "memoized" using "dp" array

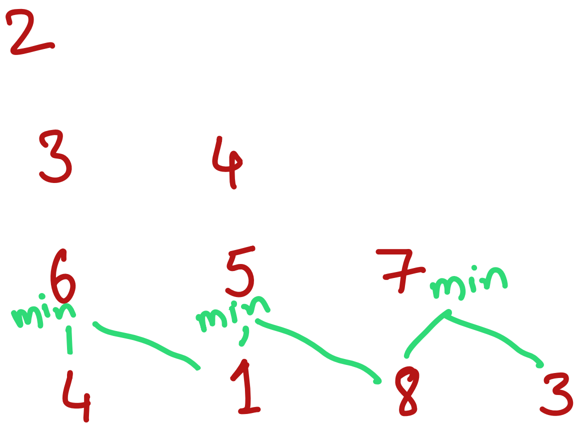
```
if( i == triangle.size()-1)
    return triangle.get(i).get(j)
if (dp[i][j] != -1) return dp[i][j]

int down = currentVal + recursion(i+1,j
int diag = current Val + recursion (i+1, j+1

return dp[i][j]= min (down , diag)
```

Memoization

Tabulation:



Tabulation is reverse of memoization. In memoization we traversed from top-bottom, here we will do reverse.

Create a list called "belowRow" and fill in the last row details.

→ Traverse from "n-2" row

→ Store the subproblem computations in a new LinkedList called "currentRow"

→ Find the cost at each "col" in the current row

→ When we traverse through the top row, only one element remains. i.e the min cost of the path to reach from first row to last Row.

```
private int tabulationSpaceOptimisedHelper(List<List<Integer>>
triangle) {
        int n = triangle.size();
        List<Integer> belowRow = triangle.get(n-1);

        for(int row = n-2;row >= 0;row--) {
            LinkedList<Integer> currentRow = new LinkedList<>();
            for(int col = row;col >= 0;col--) {
                int currentVal = triangle.get(row).get(col);
                int down = currentVal + belowRow.get(col);
                int diag = currentVal + belowRow.get(col+1);

                currentRow.addFirst(Math.min(down, diag));
            }
            belowRow = new ArrayList<>(currentRow);
        }
```