In [ ]:
```
https://leetcode.com/problems/minimum-path-sum/
```

In [ ]:
```
```

Backtracking: Brute Force

```
class Solution {
public:
    int minPathSum(vector<vector<int>>& grid) {
        if (grid.size() == 0) return 0;
        if (grid[0].size() == 0) return 0;

        return minPathSumUtil(grid, 0, 0);
    }
    int minPathSumUtil(vector<vector<int>>& grid, int i, int j) {
        if (i >= grid.size() || j >= grid[0].size()) return -1;
        if (i == grid.size() - 1 && j == grid[i].size() -1) return gr
id[i][j];

        int down = minPathSumUtil(grid, i+1, j);
        int right = minPathSumUtil(grid, i, j+1);

        if (down == -1 && right == -1) return -1;
        else if (right == -1 || down == -1) return grid[i][j] + max(d
own, right);
        else return grid[i][j] + min(down, right);
    }
};
```

```
class Solution {
public:
    int minPathSum(vector<vector<int>>& grid) {
        if (grid.size() == 0) return 0;
        if (grid[0].size() == 0) return 0;

        return minPathSumUtil(grid, 0, 0);
    }
    int minPathSumUtil(vector<vector<int>>& grid, int i, int j) {
        if (i >= grid.size() || j >= grid[0].size()) return 10000; //
number larger than given max value in grid

        if (i == grid.size() - 1 && j == grid[i].size() -1) return gr
id[i][j];

        int down = minPathSumUtil(grid, i+1, j);
        int right = minPathSumUtil(grid, i, j+1);

        return grid[i][j] + min(down, right);
    }
};
```

```
  0 1 2
0 1 3 1
1 1 5 1
2 4 2 1
```

In [ ]:

Recursive solutions

Not yet working

```
class Solution {
public:

    int min(int x, int y, int z) {
        if (x < y)
            return (x < z) ? x : z;
        else
            return (y < z) ? y : z;
    }

    int minCost(vector<vector<int>>& grid, int dest_i, int dest_j) {
        if (dest_j < 0 || dest_i < 0)
            return INT_MAX;
        else if (dest_i == 0 && dest_j == 0)
            return grid[dest_i][dest_j];
        else
            return grid[dest_i][dest_j] + min(minCost(grid, dest_i -
1, dest_j), minCost(grid, dest_i, dest_j - 1));
    }

    int minPathSum(vector<vector<int>>& grid) {
        return minCost(grid, grid.size(), grid[0].size());
    }
};
```

```
class Solution {
    int sum=0;
    public int minPathSum(int[][] grid) {
       return  minPath(grid,0,0);
    }
    public int minPath(int grid[][],int i,int j)
    {
        if(i>=grid.length||j>=grid[0].length)
            return Integer.MAX_VALUE;
        if(i==grid.length-1&&j==grid[0].length-1)
            return grid[i][j];
        int right=minPath(grid,i,j+1);
        int dwn=minPath(grid,i+1,j);
         return grid[i][j]+Math.min(right,dwn) ;
    }
}
```

```
public int rec( int [][] grid , Integer [][] dp , int i, int j ){
        if(i == grid.length && j == grid[0].length){
            return grid[i][j];
        }

        if(dp[i][j] != null)
            return dp[i][j];

        int rightSum = -1 , downSum = -1;
        if(j+1 < grid[0].length)
            rightSum = rec(grid,dp , i , j+1);
        if(i+1 < grid.length)
            downSum = rec(grid,dp,i+1,j);

        int ans = grid[i][j];
        if(rightSum != -1 && downSum != -1){
            ans +=  Math.min(rightSum , downSum);
        } else if(rightSum != -1){
           ans +=  rightSum;
        } else if(downSum != -1) {
            ans += downSum;
        }

        return dp[i][j] = ans;
    }
```

```
public int rec( int [][] grid , Integer [][] dp , int i, int j ){
        if(i == grid.length-1 && j == grid[0].length-1){
            return grid[i][j];
        }

        if(dp[i][j] != null)
            return dp[i][j];

        if(i == grid.length-1)
            return dp[i][j] = grid[i][j] + rec(grid,dp,i,j+1);
        if(j == grid[0].length-1){
            return dp[i][j] = grid[i][j] + rec(grid,dp,i+1,j);
        }
         return dp[i][j] = grid[i][j]+Math.min(rec(grid,dp,i,j+1),rec
(grid,dp,i+1,j)) ;
    }
```

In [ ]:

Tabulation

```
public int minPathSum(int[][] grid) {
        int r = grid.length;
        int c = grid[0].length;
        for(int i = 1 ; i < c; i++){
            grid[0][i] += grid[0][i-1];
        }
        for(int i = 1 ; i < r ; i++){
            grid[i][0] += grid[i-1][0];
        }
        for(int i = 1; i < r ; i++){
            for(int j = 1 ; j < c; j++){
                grid[i][j] += Math.min(grid[i-1][j] , grid[i][j-1]);
            }
        }
        return grid[r-1][c-1];
    }
```

```
class Solution {
public:
    int minPathSum(vector<vector<int>>& grid) {
        int m = grid.size();
        int n = grid[0].size();

        vector<vector<int>> res(m, vector<int>(n, -1));

        for (int i=0; i<m; i++) {
            for (int j=0; j<n; j++) {
                if (i==0 && j==0)
                    res[i][j] = grid[i][j];
                else {
                    int up = INT_MAX, left = INT_MAX;
                    if (i>0)
                        up = res[i-1][j];

                    if (j>0)
                        left = res[i][j-1];

                    res[i][j] = min(up, left) + grid[i][j];
                }
            }
        }

        return res[m-1][n-1];
    }
};
```

In [ ]:

In [ ]:
```
https://leetcode.com/problems/house-robber/description/
```

```
class Solution {
  public int rob(int[] nums) {
    int n = nums.length;
    if (n == 0)
      return 0;
    if (n == 1)
      return nums[0];
    int[] max_val = new int[n];
    max_val[0] = nums[0];
    max_val[1] = Math.max(nums[0], nums[1]);

    for (int i = 2; i < n; i++)
      max_val[i] = Math.max(max_val[i - 1], max_val[i - 2] + nums
[i]);

    return max_val[n - 1];
  }
}



class Solution {
public:
    int rob(vector<int>& nums) {
        int n = nums.size();

        if (n == 1)
            return nums[0];

        vector<int> res(n, INT_MIN);
        res[0] = nums[0];
        res[1] = max(nums[0], nums[1]);

        for (int i=2; i<n; i++) {
            res[i] = max(res[i-1], res[i-2]+nums[i]);
        }

        return res[n-1];
    }
};
```

```
class Solution {
    public int rec(int [] nums, int i ,Integer dp[]){
        int n = nums.length;
        int ans = nums[i];
        if(dp[i] != null)
            return dp[i];
        if(i+2 < n){
            ans += rec(nums,i+2,dp);
        }
        if(i+3 < n){
            ans = Math.max(ans, nums[i] + rec(nums,i+3,dp));
        }
        return dp[i] = ans;
    }
    public int rob(int[] nums) {
        int n = nums.length;
        Integer[] dp = new Integer[n];
        if(n==1)
            return nums[0];
        return Math.max(rec(nums,0,dp) , rec(nums,1,dp));
    }
}
```

In [ ]:

In [ ]:
```
Number of ways to make N as sum of 1,3,5. [Permutations]
```

In [ ]:
```
6
[1 1 1 1 1 1]
[1 3 1 1]
[1 1 3 1]
[1 1 1 3]
[3 1 1 1]
[3 3]
[1 5]
[5 1]
```

In [ ]:

In [ ]:
```
- Count, min, max

DP:
1. memoization : recursion + cache : n-d array or hash map
2. tabulation : iterative + cache: n-d array
```

In [ ]:
```
- Recurrence relation
- How many variables determine the current state/solution
```

backtracking: brute force

In [16]:
```
# 1, 3, 5
def way(n):
    if n < 0:
        return 0
    if n == 0:
        return 1

    return way(n-1) + way(n-3) + way(n-5)
#                2          0            -2
#                (1)       (1)
print(way(2)) # [1,1]
print(way(3)) # [1,1,1] [3]
print(way(6))
print(way(20))
print(way(30))
print(way(40))
```

```
1
2
8
4285
390257
35543051
```

In [ ]:

Memoisation

In [11]:
```
# 1, 3, 5
import functools

@functools.lru_cache
def way(n):
    if n < 0:
        return 0
    if n == 0  or n == 1:
        return 1

    return way(n-1) + way(n-3) + way(n-5)
#               2          0            -2
#               (1)       (1)
print(way(2)) # [1,1]
print(way(3)) # [1,1,1] [3]
print(way(6))
print(way(20))
print(way(30))
print(way(40))
print(way(50))
```

```
1
2
8
4285
390257
35543051
3237119305
```

In [12]:
```
# 1, 3, 5

cache = {}

def way(n):
    if n < 0:
        return 0
    if n == 0  or n == 1:
        return 1

    if n in cache:
        return cache[n]

    cache[n] =  way(n-1) + way(n-3) + way(n-5)
    return cache[n]
#               2          0            -2
#               (1)       (1)
print(way(2)) # [1,1]
print(way(3)) # [1,1,1] [3]
print(way(6))
print(way(20))
print(way(30))
print(way(40))
print(way(50))
```

```
1
2
8
4285
390257
35543051
3237119305
```

In [19]:

```python
cache = {}
def way(n, nums):
    if n < 0:
        return 0
    if n == 0:
        return 1

    if n in cache:
        return cache[n]

    count = 0
    for num in nums:
        count += way(n-num, nums)

    cache[n] = count
    return count


print(way(2, [1,3,5])) # [1,1]
print(way(3,[1,3,5]))
print(way(6,[1,3,5]))
print(way(20,[1,3,5]))
print(way(30,[1,3,5]))
print(way(40,[1,3,5]))
print(way(50,[1,3,5]))
```

```
1
2
8
4285
390257
35543051
3237119305
```

Tabulation

```
public int sol( int n ){
        int [] dp = new int [n+1];
        dp[0] = 1;
        int [] val = new int [] { 1,3, 5};
        for(int i = 1 ; i <= n ; i++){
                for(int j = 0 ;j < 3; j++){
                    if(i-val[j] >= 0)
                        dp[i] += dp[i-val[j]];

                }

            }

        return dp[n];
    }



def numberOfWays(n):
    cache = [0]*(n + 1)
    cache[0] = 1

    for i in range(1, n+1):
        f1 = cache[i-1] if i-1 >= 0 else 0
        f2 = cache[i-3] if i-3 >= 0 else 0
        f3 = cache[i-5] if i-5 >= 0 else 0

        cache[i] = f1 + f2 + f3


    return cache[-1]



print(numberOfWays(2))
print(numberOfWays(3))
print(numberOfWays(40))
print(numberOfWays(50))
```

```
int  solve(int val, vector<int> & nums)
{
    vector<int> res(val+1,1);
    res[0]=1;
 for(int i=1;i<=(val);i++)
 {
        for(int j=0;j<nums.size();j++)
        {
            if (i-nums[j]>0 )
            res[i]+=res[i-nums[j]];
        }
    }
    return res[val];
}
int main() {
    vector<int> data{1,3,5};
   std::cout<< solve(3,data);
}
```

In [ ]:

In [39]:
```
# ["aabb", "aaaa", "bbab"]

# "abc" "aaaa" "aa" "bca", "bcaaa", "aabc", "aaabc"


 # bcaaa  aaaa aa  abc


# a: {"s": 2    2    2 max
#     "e":0     0    0 max
#      "m":0}   4    4 sum

# b: {"s": 0    0    2
#     "e":2     2    2
#      "m":0}   0    0

# s + m + e

# 6?

def eval(word):
    s = word[0]
    e = word[-1]

    s_count = 0
    for c in word:
        if c == s:
            s_count+=1
        else:
            break

    e_count = 0
    for c in reversed(word):
        if c == e:
            e_count+=1
        else:
            break

    res = [ (s, ('s', s_count)), (e, ('e', e_count)) ]
    if len(word) == word.count(s):
        res = [ (s, ('m', len(word))) ]

    return res

def concat(words):

    counts = {}
    for word in words: # total size of all strings

        curr = eval(word) # len(len)
        print(curr)
        for k,v in curr:

            if k not in counts:
                counts[k] = {'s':0, 'e':0, 'm':0}
            if v[0] == 'm':
                counts[k]['m'] += v[1]
            else:
                counts[k][v[0]] = max(counts[k][v[0]], v[1])

    max_count = 0
    for _, count in counts.items():
        print(count)
        max_count = max( sum([v for _,v in count.items()]), max_count )

    return max_count


concat(["aacdadbb", "aaaa", "bbab"])
concat(["aacdadbb", "aaaa", "a", "bbab"])
concat(["aacdadbb", "aaaa", "a", "bbab", "bbbbbbbbbbbbbbba"])

```

```
[('a', ('s', 2)), ('b', ('e', 2))]
[('a', ('m', 4))]
[('b', ('s', 2)), ('b', ('e', 1))]
{'s': 2, 'e': 0, 'm': 4}
{'s': 2, 'e': 2, 'm': 0}
[('a', ('s', 2)), ('b', ('e', 2))]
[('a', ('m', 4))]
[('a', ('m', 1))]
[('b', ('s', 2)), ('b', ('e', 1))]
{'s': 2, 'e': 0, 'm': 5}
{'s': 2, 'e': 2, 'm': 0}
[('a', ('s', 2)), ('b', ('e', 2))]
[('a', ('m', 4))]
[('a', ('m', 1))]
[('b', ('s', 2)), ('b', ('e', 1))]
[('b', ('s', 15)), ('a', ('e', 1))]
{'s': 2, 'e': 1, 'm': 5}
{'s': 15, 'e': 2, 'm': 0}
```

Out[39]: 17

In [ ]:

```
import java.util.*;

public class ContactStringFromArray {
    public static void main(String[] args) {
         System.out.println(longestSingleCharSubstring(new String[]
{"aabb","aaaa", "bbab"}));
    }
    public static int longestSingleCharSubstring(String[] arr) {
        List <String> alistofAllCombination =new ArrayList<>();
        generateCombinationsHelper(arr, "", 0, alistofAllCombinatio
n);
        return longestCharSeq(alistofAllCombination);
    }

    public static void generateCombinationsHelper(String[] arr, Strin
g currentCombination, int currentIndex, List<String> allCombinations)
{
        if (currentIndex == arr.length) {
            allCombinations.add(currentCombination);
            return;
        }
        for (int i = 0; i < arr.length; i++) {
            generateCombinationsHelper(arr, currentCombination + arr
[i], currentIndex + 1, allCombinations);
        }
    }
    public static int longestCharSeq(List <String> str) {
        HashMap <String,Integer> hmap = new HashMap<>();
        for(String s : str) {
            hmap.put(s,longestCharSeqCountHelper(Collections.singleto
nList(s).toString()));
        }
       return longestCharSeqOccur(hmap);
    }

    private static int longestCharSeqOccur(HashMap<String, Integer> h
map) {
        Optional<Map.Entry<String, Integer>> maxEntry = hmap.entrySet
()
                .stream()
                .max(Map.Entry.comparingByValue());
        return maxEntry.get()
                .getValue();
    }

    public static int longestCharSeqCountHelper(String s) {
        int maxCount = 1;
```

```
                char c = s.charAt(0);
                int count = 1;

                for (int i = 1; i < s.length(); i++) {
                    if (c == s.charAt(i)) {
                        count++;
                        maxCount = Math.max(maxCount, count);
                    } else {
                        c = s.charAt(i);
                        count = 1;
                    }
                }
                return maxCount;
            }
        }
```