***Week – 5 (26.04.2021 – 02.05.2021)***

***RANDOM CODES***

1. ***Add Strings:***

class Solution {

public:

string addStrings(string num1, string num2) {

int sum = 0, carry = 0, i = num1.size()-1, j = num2.size()-1;

string res = "";

while(i>=0 || j>=0)

{

sum = carry;

carry = 0;

if(i>=0)

{

sum = sum + num1[i] - '0';

i--;

}

if(j>=0)

{

sum = sum + num2[j] - '0';

j--;

}

carry = sum / 10;

sum = sum % 10;

res = to\_string(sum) + res;

}

if(carry!=0) res = to\_string(carry) + res;

return res;

}

};

1. ***Add to Array-Form of Integer:***

class Solution {

public:

vector<int> addToArrayForm(vector<int>& num, int k) {

int i = num.size()-1, sum = 0, carry = 0;

vector<int> result;

while(i>=0 || k>0)

{

sum = carry;

carry = 0;

if(k>0)

{

sum = sum + (k%10);

k = k/10;

}

if(i>=0)

{

sum = sum + num[i];

i--;

}

carry = sum/10;

sum = sum%10;

result.push\_back(sum);

}

if(carry!=0)

{

while(carry>0)

{

sum = carry%10;

result.push\_back(sum);

carry = carry/10;

}

}

reverse(result.begin(), result.end());

return result;

}

};

1. ***First Unique Character in a String:***

class Solution {

public:

int firstUniqChar(string s) {

int fre[26] = {0}, i;

for(i=0; i<s.size(); i++)

fre[s[i] - 'a']++;

for(i=0; i<s.size(); i++)

if(fre[s[i] - 'a'] == 1) return i;

return -1;

}

};

1. ***Sum of Square Numbers:***

class Solution {

public:

bool judgeSquareSum(int c) {

int top = sqrt(c), i = 0, x;

while(i<=top)

{

x = i\*i;

if(x == c - top\*top) return true;

if(x > c - top\*top) top--;

if(x < c - top\*top) i++;

}

return false;

}

};

1. ***Valid Square:***

class Solution {

public:

bool validSquare(vector<int>& p1, vector<int>& p2, vector<int>& p3, vector<int>& p4) {

vector<int> res(6);

int x, i;

res[0] = (pow((p1[0]-p2[0]),2) + pow((p1[1]-p2[1]),2));

res[1] = (pow((p1[0]-p3[0]),2) + pow((p1[1]-p3[1]),2));

res[2] = (pow((p1[0]-p4[0]),2) + pow((p1[1]-p4[1]),2));

res[3] = (pow((p2[0]-p3[0]),2) + pow((p2[1]-p3[1]),2));

res[4] = (pow((p2[0]-p4[0]),2) + pow((p2[1]-p4[1]),2));

res[5] = (pow((p3[0]-p4[0]),2) + pow((p3[1]-p4[1]),2));

sort(res.begin(), res.end());

x = res[0];

if(x == 0 || res[1]!=x || res[2]!=x || res[3]!=x || res[4]!=2\*x || res[5]!=2\*x)

return false;

return true;

}

};

1. ***Number of Segments in a String:***

class Solution {

public:

int countSegments(string s) {

if(s.size() == 0) return 0;

int i, count=0;

for(i=0; i<s.size(); i++)

if((i == 0 || s[i-1] == ' ') && s[i] != ' ')

count++;

return count;

}

};

1. ***Count Numbers with Unique Digits:***

class Solution {

public:

int countNumbersWithUniqueDigits(int n) {

int ans = 1;

if(n>=1) ans +=9;

if(n>=2) ans+= 9\*9;

if(n>=3) ans+=9\*9\*8;

if(n>=4) ans+=9\*9\*8\*7;

if(n>=5) ans+=9\*9\*8\*7\*6;

if(n>=6) ans+=9\*9\*8\*7\*6\*5;

if(n>=7) ans+=9\*9\*8\*7\*6\*5\*4;

if(n>=8) ans+=9\*9\*8\*7\*6\*5\*4\*3;

return ans;

}

};

1. Nth Digit:

class Solution {

public:

int findNthDigit(int n) {

if(n < 10) return n;

int64\_t x{9}, count{1};

while(n-x\*count > 0) {

n -= x\*count;

++count;

x \*= 10;

}

return (int)(pow(10, count-1) + ((n-1)/count))/(int)pow(10, (count - (n % count)) % count) % 10;

}

};

1. ***Integer Replacement:***

class Solution {

public:

int integerReplacement(long long int n) {

int count = 0;

while(n!=1)

{

if(n%2 == 1)

{

count++;

if(((n-1)/2)%2==0||(n-1)/2==1) n-=1;

else n++;

}

count++;

n = n/2;

}

return count;

}

};

1. ***The kth Factor of n:***

class Solution {

public:

int kthFactor(int n, int k) {

int i, count = 0;

for(i=1; i<=n; i++)

if(n%i == 0)

{

count++;

if(count == k) return i;

}

return -1;

}

};

1. ***Largest Perimeter Triangle:***

class Solution {

public:

int largestPerimeter(vector<int>& nums) {

sort(nums.begin(), nums.end());

int i;

for(i=nums.size()-3; i>=0; i--)

if(nums[i] + nums[i+1] > nums[i+2])

return (nums[i] + nums[i+1] + nums[i+2]);

return 0;

}

};

1. ***Largest Triangle Area:***

class Solution {

public:

double largestTriangleArea(vector<vector<int>>& points) {

double ans = 0;

int i, j, k;

for(i=0; i<points.size(); i++)

for(j=i+1; j<points.size(); j++)

for(k=j+1; k<points.size(); k++)

{

double a = sqrt(pow(points[i][0] - points[j][0], 2) + pow(points[i][1] - points[j][1], 2)),

b = sqrt(pow(points[j][0] - points[k][0], 2) + pow(points[j][1] - points[k][1], 2)),

c = sqrt(pow(points[k][0] - points[i][0], 2) + pow(points[k][1] - points[i][1], 2)),

s = (a + b + c)/2;

ans = max(ans, sqrt(s\*(s-a)\*(s-b)\*(s-c)));

}

return ans;

}

};

1. ***Valid Boomerang:***

class Solution {

public:

bool isBoomerang(vector<vector<int>>& points) {

return (points[2][0]-points[0][0])\*(points[1][1]-points[0][1]) != (points[2][1]-points[0][1])\*(points[1][0]-points[0][0]);

}

};

1. ***Angle Between Hands of a Clock:***

class Solution {

public:

double angleClock(int hour, int minutes) {

if(hour < 0 || minutes < 0 || hour > 12 || minutes > 60)

return 0.0;

if(hour == 12) hour = 0;

double hoursAngle = (hour \* 60 + minutes) \* 0.5;

double minutesAngle = minutes \* 6;

double angle = abs(hoursAngle - minutesAngle);

angle = min(360 - angle, angle);

return angle;

}

};

1. ***Sign of the Product of an Array:***

class Solution {

public:

int arraySign(vector<int>& nums) {

int count = 0, i;

for(i=0; i<nums.size(); i++)

{

if(nums[i]<0) count++;

if(nums[i] == 0) return 0;

}

if(count%2 == 0) return 1;

else return -1;

}

};