**Sum of squares of first n even numbers**

[2n(n + 1)(2n + 1)] / 3

**Sum of squares of first n odd numbers**

[n(2n+1)(2n-1)] / 3

**Sum of Co-prime Numbers of an Integer**

**A math equation with numbers and symbols

Description automatically generated**

**Max subarray xor using Trie**

**const int N = 2;**

**struct node{**

**node\* arr[N];**

**};**

**node\* getNode()**

**{**

**node\* root = new node();**

**root->arr[0] = NULL;**

**root->arr[1] = NULL;**

**return root;**

**}**

**void insert(node\* root, int n)**

**{**

**node \*tempRoot = root;**

**for(int i = 31; i >= 0; i--)**

**{**

**int index = ((n >> i) & 1);**

**if(tempRoot->arr[index] == NULL)**

**{**

**tempRoot->arr[index] = getNode();**

**}**

**tempRoot = tempRoot->arr[index];**

**}}**

**int search(node\* root, int n)**

**{**

**node\* tempRoot = root;**

**int res = 0;**

**for(int i = 31; i >= 0; i--){**

**int index = ((n>>i)&1);**

**if(tempRoot->arr[index^1]){**

**res += (1 << i);**

**tempRoot = tempRoot->arr[index^1];**

**}else{**

**tempRoot = tempRoot->arr[index];**

**}}**

**return res;**

**}**

**void deleteTrie(node \*root)**

**{**

**for(int i = 0; i < N; i++){**

**if(root->arr[i]){**

**deleteTrie(root->arr[i]);**

**}}**

**delete root;**

**}**

**void solve(){**

**node\* root = getNode();**

**insert(root,0);**

**int n;**

**cin >> n;**

**vector<int> v(n);**

**int pxor = 0;**

**int mxor = 0;**

**for(int i = 0; i < n; i++)**

**{**

**cin >> v[i];**

**pxor ^= v[i];**

**mxor = max(mxor,pxor);**

**mxor = max(mxor,search(root,pxor));**

**insert(root,pxor);**

**}**

**deleteTrie(root);**

**cout << mxor << endl;**

**}**

**Max subarray xor 2d using Trie**

**int trie[10001\*28][2],node;**

**void insert(int n){**

**int root = 1;**

**for(int i = 27; i >= 0; i--){**

**int idx = (1 & (n >> i));**

**if(!trie[root][idx]){**

**trie[root][idx] = node++;**

**}**

**root = trie[root][idx];**

**}}**

**int search(int n){**

**int root = 1;**

**int res = 0;**

**for(int i = 27; i >= 0; i--){**

**int idx = (1 & (n >> i));**

**if(trie[root][idx^1]){**

**res += (1 << i);**

**root = trie[root][idx^1];**

**}else {**

**root = trie[root][idx];**

**}}**

**return res;**

**}**

**void solve(){**

**int n, m;**

**cin >> n >> m;**

**int v[n + 5][m + 5];**

**for (int i = 0; i <= n; i++){**

**v[i][0] = 0;**

**}**

**for (int i = 1; i <= n; i++){**

**for (int j = 1; j <= m; j++){**

**cin >> v[i][j];**

**v[i][j] ^= v[i][j - 1];**

**}}**

**int mxor = 0;**

**for (int l = 1; l <= m; l++){**

**for (int r = l; r <= m; r++){**

**memset(trie,0,sizeof trie);**

**node = 2;**

**int rowXor = 0;**

**insert(0);**

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

**{**

**rowXor = (rowXor ^ v[i][r] ^ v[i][l - 1]);**

**int k = search(rowXor);**

**mxor = max(mxor, rowXor);**

**mxor = max(mxor, k);**

**insert(rowXor);**

**}}}**

**cout << mxor << endl;**

**}**

**Sparse table**

**const int N = 1e5+100;**

**int t[100][N];**

**int it[100][N];**

**int Log2[N];**

**int n;**

**int p;**

**void init()**

**{**

**for(int i = 2; i <= n; i++)  // storing log values**

**{**

**Log2[i] = Log2[i/2]+1;**

**}**

**p = Log2[n];**

**for(int i = 0; i < n; i++) it[0][i] = i; // init idx**

**for(int i = 1; i <= p; i++)  { // for max**

**for(int j = 0; j+(1<<i) <= n; j++){**

**int left = t[i-1][j];**

**int right = t[i-1][j+(1<<(i-1))];**

**t[i][j] = max(left,right);**

**if(left >= right){**

**it[i][j] = it[i-1][j];**

**}else{**

**it[i][j] = it[i-1][j+(1<<i-1)];**

**}}}}**

**int idx = -1;**

**int query(int l, int r) // TC: O(1) {**

**int len = r-l+1;**

**int p = Log2[len];**

**int left = t[p][l];**

**int right = t[p][r-(1<<p)+1];**

**if(left >= right){**

**idx = it[p][l];**

**}else**

**{**

**idx = it[p][r-(1<<p)+1];**

**}**

**return max(left,right);**

**}**

**int overlapQuery(int l, int r){**

**int mx = INT\_MIN;**

**for(int p = Log2[r-l+1]; l <= r; p = Log2[r-l+1]){**

**mx = max(mx,t[p][l]);**

**l += (1<<p);**

**}**

**return mx;**

**}**

**int main(){**

**cin >> n;**

**for(int i = 0; i < n; i++){**

**cin >> t[0][i];**

**}**

**int q;**

**cin >> q;**

**init();**

**while(q--){**

**int l, r;**

**cin >> l >> r;**

**int val = query(l,r);**

**cout << "idx : " << idx << endl;**

**cout << "val : " << val << endl;**

**cout << "overlap : " << overlapQuery(l,r) << endl;**

**}}**

**/\***

**13**

**4 2 3 7 1 5 3 3 9 6 7 -1 4**

**100**

**\*/**

**Digit DP: Number of zeroes**

**#include <bits/stdc++.h>**

**using namespace std;**

**int dp1[11][2][2];**

**int dp2[11][2];**

**int ara[20];**

**int func2(int pos, int isSmall){**

**if(pos == 10) return 1;**

**if(dp2[pos][isSmall] != -1) return dp2[pos][isSmall];**

**int lo = 0, hi = ara[pos], re = 0;**

**if(isSmall) hi = 9;**

**for(int i = lo; i <= hi; i++){**

**re += func2(pos+1,isSmall | (i < hi));**

**}**

**return dp2[pos][isSmall] = re;**

**}**

**int func(int pos, int isSmall, int hasStarted){**

**if(pos == 10) return 0;**

**if(dp1[pos][isSmall][hasStarted] != -1) return dp1[pos][isSmall][hasStarted];**

**int lo = 0, hi = ara[pos], re = 0;**

**if(isSmall) hi = 9;**

**for(int i = lo; i <= hi; i++){**

**int val = func(pos+1, isSmall | (i < hi), hasStarted | (i != 0));**

**if(hasStarted && i == 0) val = val + func2(pos+1, isSmall | i < hi);**

**re += val;**

**}**

**return dp1[pos][isSmall][hasStarted] = re;**

**}**

**int main(){**

**string str;**

**cin >> str;**

**int k = 10-str.size();**

**for(int i = 0;i < str.size();i++) ara[k++] = str[i]-'0';**

**memset(dp1,-1,sizeof(dp1));**

**memset(dp2,-1,sizeof(dp2));**

**cout << func(1,0,0) << endl;**

**}**

**Digit DP: Digit Sum**

**#include <bits/stdc++.h>**

**using namespace std;**

**string str;**

**int dp[12][92][2];**

**int f(int pos, int n, int sum,int flag)**

**{**

**if(pos > n) return sum;**

**if(dp[pos][sum][flag] != -1) return dp[pos][sum][flag];**

**int limit = 9;**

**if(flag == false) limit = (str[pos-1]-'0');**

**int total = 0,k = 0;**

**for(int i = 0; i <= limit; i++)**

**{**

**if(flag || i < limit)**

**{**

**k = f(pos+1,n,sum+i,true);**

**total += k;**

**}else**

**{**

**k = f(pos+1,n,sum+i,false);**

**total += k;**

**}**

**}**

**return dp[pos][sum][flag] = total;**

**}**

**int main()**

**{**

**memset(dp,-1,sizeof dp);**

**cin >> str;**

**cout << f(1,str.size(),0,false) << endl;**

**}**