|  |
| --- |
| Chinese Remainder Theorem.cpp |
| #include<bits/stdc++.h> |
|  |
| using namespace std; |
|  |
| const int MAXP = 9765625 + 1; |
| const long long MOD = 10000000000LL; |
|  |
| namespace crt{ |
| long long extended\_gcd(long long a, long long b, long long& x, long long& y){ |
| if (!b){ |
| y = 0, x = 1; |
| return a; |
| } |
| long long g = extended\_gcd(b, a % b, y, x); |
| y -= ((a / b) \* x); |
| return g; |
| } |
|  |
| long long mod\_inverse(long long a, long long m){ |
| long long x, y; |
| extended\_gcd(a, m, x, y); |
| return (x + m) % m; |
| } |
|  |
| long long chinese\_remainder(vector <long long> ar, vector <long long> mods){ |
| long long x, y, res = 0, M = 1; |
| for (int i = 0; i < ar.size(); i++) M \*= mods[i]; |
| for (int i = 0; i < ar.size(); i++){ |
| x = M / mods[i], y = mod\_inverse(x, mods[i]); |
| res = (res + (((x \* ar[i]) % M) \* y)) % M; |
| } |
| return res; |
| } |
| } |

|  |
| --- |
| Modular Exponentiation.cpp |
| long long BigMod(long long B,long long P,long long M) |
| { |
| long long R=1; |
| while(P>0) |
| { |
| if(P%2==1) |
| { |
| R=(R\*B)%M; |
| } |
| P/=2; |
| B=(B\*B)%M; |
| } |
| return R; |
| } |

|  |
| --- |
| Prime Counting Function.cpp |
| #include<bits/stdc++.h> |
|  |
| using namespace std; |
|  |
| #define fRead(x) freopen(x,"r",stdin) |
| #define fWrite(x) freopen (x,"w",stdout) |
|  |
| #define LL long long |
| #define ULL unsigned long long |
| #define ff first |
| #define ss second |
| #define pb push\_back |
| #define PI acos(-1.0) |
| #define mk make\_pair |
| #define pii pair<int,int> |
| #define pll pair<LL,LL> |
|  |
|  |
| #define min3(a,b,c) min(a,min(b,c)) |
| #define max3(a,b,c) max(a,max(b,c)) |
| #define min4(a,b,c,d) min(a,min(b,min(c,d))) |
| #define max4(a,b,c,d) max(a,max(b,max(c,d))) |
| #define SQR(a) ((a)\*(a)) |
| #define FOR(i,a,b) for(int i=a;i<=b;i++) |
| #define ROF(i,a,b) for(int i=a;i>=b;i--) |
| #define REP(i,b) for(int i=0;i<b;i++) |
| #define MEM(a,x) memset(a,x,sizeof(a)) |
| #define ABS(x) ((x)<0?-(x):(x)) |
|  |
| #define SORT(v) sort(v.begin(),v.end()) |
| #define REV(v) reverse(v.begin(),v.end()) |
| //#pragma GCC target ("avx2") |
| //#pragma GCC optimization ("O3") |
| //#pragma GCC optimization ("unroll-loops") |
| //#pragma comment(linker, "/stack:200000000") |
| //#pragma GCC optimize("Ofast") |
| //#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native") |
| #define FastRead ios\_base::sync\_with\_stdio(0);cin.tie(nullptr); |
| #ifdef VAMP |
| #define debug(...) \_\_f(#\_\_VA\_ARGS\_\_, \_\_VA\_ARGS\_\_) |
| template < typename Arg1 > |
| void \_\_f(const char\* name, Arg1&& arg1){ |
| cout << name << " = " << arg1 << std::endl; |
| } |
| template < typename Arg1, typename... Args> |
| void \_\_f(const char\* names, Arg1&& arg1, Args&&... args){ |
| const char\* comma = strchr(names+1, ','); |
| cout.write(names, comma - names) << " = " << arg1 <<" | "; |
| \_\_f(comma+1, args...); |
| } |
| #else |
| #define debug(...) |
| #endif |
|  |
| namespace pcf{ |
| /// Prime-Counting Function |
| /// initialize once by calling init() |
| /// Legendre(n) and Lehmer(n) returns the number of primes less than or equal to n |
| /// Lehmer(n) is faster |
|  |
| #define MAXN 1000010 /// initial sieve limit |
| #define MAX\_PRIMES 1000010 /// max size of the prime array for sieve |
| #define PHI\_N 100000 |
| #define PHI\_K 100 |
|  |
| unsigned int ar[(MAXN >> 6) + 5] = {0}; |
| int len = 0; /// total number of primes generated by sieve |
| int primes[MAX\_PRIMES]; |
| int counter[MAXN]; /// counter[m] --> number of primes <= i |
| int phi\_dp[PHI\_N][PHI\_K]; /// precal of phi(n,k) |
|  |
| bitset <MAXN> isComp; |
| //bool isComp[MAXN]; |
| void Sieve(int N){ |
| int i,j,sq = sqrt(N); |
| isComp[1] = true; |
| for(i=4;i<=N;i+=2) isComp[i] = true; |
| for(i=3;i<=sq;i+=2){ |
| if(!isComp[i]){ |
| for(j=i\*i;j<=N;j+=i+i) isComp[j] = 1; |
| } |
| } |
| for (i = 1; i <= N; i++){ |
| if (!isComp[i]) primes[len++] = i; |
| counter[i] = len; |
| } |
| } |
|  |
| void init(){ |
| Sieve(MAXN - 1); |
|  |
| /// precalculation of phi upto size (PHI\_N,PHI\_K) |
| int k , n , res; |
| for(n = 0; n < PHI\_N; n++) phi\_dp[n][0] = n; |
| for (k = 1; k < PHI\_K; k++){ |
| for (n = 0; n < PHI\_N; n++){ |
| phi\_dp[n][k] = phi\_dp[n][k - 1] - phi\_dp[n / primes[k - 1]][k - 1]; |
| } |
| } |
| } |
|  |
| /// returns number of integers less or equal n which are |
| /// not divisible by any of the first k primes |
| /// recurrence --> phi( n , k ) = phi( n , k-1 ) - phi( n / p\_k , k-1) |
| long long phi(long long n, int k){ |
| if (n < PHI\_N && k < PHI\_K) return phi\_dp[n][k]; |
| if (k == 1) return ((++n) >> 1); |
| if (primes[k - 1] >= n) return 1; |
| return phi(n, k - 1) - phi(n / primes[k - 1], k - 1); |
| } |
|  |
|  |
| long long Legendre(long long n){ |
| if (n < MAXN) return counter[n]; |
|  |
| int lim = sqrt(n) + 1; |
| int k = upper\_bound(primes, primes + len, lim) - primes; |
| return phi(n, k) + (k - 1); |
| } |
|  |
| ///complexity: n^(2/3).(logn)^(1/3) |
| long long Lehmer(long long n){ |
| if (n < MAXN) return counter[n]; |
|  |
| long long w , res = 0; |
| int i, j, a, b, c, lim; |
| b = sqrt(n), c = Lehmer(cbrt(n)), a = Lehmer(sqrt(b)), b = Lehmer(b); |
| res = phi(n, a) + (((b + a - 2) \* (b - a + 1)) >> 1); |
|  |
| for (i = a; i < b; i++){ |
| w = n / primes[i]; |
| lim = Lehmer(sqrt(w)), res -= Lehmer(w); |
|  |
| if (i <= c){ |
| for (j = i; j < lim; j++){ |
| res += j; |
| res -= Lehmer(w / primes[j]); |
| } |
| } |
| } |
| return res; |
| } |
| } |
|  |
|  |
| int main() |
| { |
| #ifdef VAMP |
| clock\_t tStart = clock(); |
| freopen("input.txt", "r", stdin); |
| freopen("output.txt", "w", stdout); |
| #endif |
|  |
| #ifdef VAMP |
| fprintf(stderr, "\n>> Runtime: %.10fs\n", (double) (clock() - tStart) / CLOCKS\_PER\_SEC); |
| #endif |
| } |

|  |
| --- |
| Sieve of Eratosthenes.cpp |
| const int N = 100000; |
| bool stat[N+5]; |
| vector<int>primes; |
| void siv() |
| { |
| int sq = sqrt(N);for(int i =4;i<N;i+=2)stat[i]=1; |
| for(int i=3;i<=sq;i+=2)if(!stat[i])for(int j=i\*i;j<N;j+=2\*i)stat[j]=1; |
| for(int i = 2;i < N;i++)if(!stat[i])primes.push\_back(i); |
| } |

|  |
| --- |
| **Merge sort in C++.cpp** |
|  |
| #include <iostream> |
| using namespace std; |
|  |
| // Merge two subarrays L and M into arr |
| void merge(int arr[], int p, int q, int r) { |
|  |
| // Create L ← A[p..q] and M ← A[q+1..r] |
| int n1 = q - p + 1; |
| int n2 = r - q; |
|  |
| int L[n1], M[n2]; |
|  |
| for (int i = 0; i < n1; i++) |
| L[i] = arr[p + i]; |
| for (int j = 0; j < n2; j++) |
| M[j] = arr[q + 1 + j]; |
|  |
| // Maintain current index of sub-arrays and main array |
| int i, j, k; |
| i = 0; |
| j = 0; |
| k = p; |
|  |
| // Until we reach either end of either L or M, pick larger among |
| // elements L and M and place them in the correct position at A[p..r] |
| while (i < n1 && j < n2) { |
| if (L[i] <= M[j]) { |
| arr[k] = L[i]; |
| i++; |
| } else { |
| arr[k] = M[j]; |
| j++; |
| } |
| k++; |
| } |
|  |
| // When we run out of elements in either L or M, |
| // pick up the remaining elements and put in A[p..r] |
| while (i < n1) { |
| arr[k] = L[i]; |
| i++; |
| k++; |
| } |
|  |
| while (j < n2) { |
| arr[k] = M[j]; |
| j++; |
| k++; |
| } |
| } |
|  |
| // Divide the array into two subarrays, sort them and merge them |
| void mergeSort(int arr[], int l, int r) { |
| if (l < r) { |
| // m is the point where the array is divided into two subarrays |
| int m = l + (r - l) / 2; |
|  |
| mergeSort(arr, l, m); |
| mergeSort(arr, m + 1, r); |
|  |
| // Merge the sorted subarrays |
| merge(arr, l, m, r); |
| } |
| } |
|  |
| // Print the array |
| void printArray(int arr[], int size) { |
| for (int i = 0; i < size; i++) |
| cout << arr[i] << " "; |
| cout << endl; |
| } |
|  |
| // Driver program |
| int main() { |
| int arr[] = {6, 5, 12, 10, 9, 1}; |
| int size = sizeof(arr) / sizeof(arr[0]); |
|  |
| mergeSort(arr, 0, size - 1); |
|  |
| cout << "Sorted array: \n"; |
| printArray(arr, size); |
| return 0; |
| } |

|  |
| --- |
| **Binary Search in C++.cpp** |
|  |
| #include <iostream> |
| using namespace std; |
|  |
| int binarySearch(int array[], int x, int low, int high) { |
| if (high >= low) { |
| int mid = low + (high - low) / 2; |
|  |
| // If found at mid, then return it |
| if (array[mid] == x) |
| return mid; |
|  |
| // Search the left half |
| if (array[mid] > x) |
| return binarySearch(array, x, low, mid - 1); |
|  |
| // Search the right half |
| return binarySearch(array, x, mid + 1, high); |
| } |
|  |
| return -1; |
| } |
|  |
| int main(void) { |
| int array[] = {3, 4, 5, 6, 7, 8, 9}; |
| int x = 4; |
| int n = sizeof(array) / sizeof(array[0]); |
| int result = binarySearch(array, x, 0, n - 1); |
| if (result == -1) |
| printf("Not found"); |
| else |
| printf("Element is found at index %d", result); |
| } |

**By mimtiaz:**

|  |
| --- |
| **Sieve.cpp** |
| #include <iostream> |
| #include <vector> |
| #define N 1000000+10 |
|  |
| using namespace std; |
|  |
| vector<bool> prime (N, true); |
| vector<int> primeSum (N, 0); |
|  |
| int digitSum(int n){ |
| if(n==0) return n; |
| return (n%10) + digitSum(n/10); |
| } |
|  |
| void sieve(int n) { |
| prime[0] = false; |
| prime[1] = false; |
|  |
| for(int i=2;i<=N;i++){ |
| primeSum[i] = primeSum[i-1]; |
| if(prime[i]){ |
| if(prime[digitSum(i)]) primeSum[i]++; |
|  |
| for(int j=i+i;j<=N;j+=i) prime[j] = false; |
| } |
| } |
| } |
|  |
| int main(){ |
| sieve(N); |
|  |
| int t; |
| scanf("%d", &t); |
| while(t--){ |
| int a,b; |
| scanf("%d%d", &a, &b); |
| printf("%d\n", primeSum[b] - primeSum[a-1]); |
| } |
| } |

|  |
| --- |
| **Running Median.cpp** |
| #include<bits/stdc++.h> |
| using namespace std; |
|  |
| int main(){ |
| priority\_queue <int> maxHeap; |
| priority\_queue <int, vector<int>, greater<int>> minHeap; |
|  |
| int n; |
| cin >> n; |
|  |
| while(n--){ |
| int temp; |
| cin >> temp; |
|  |
| minHeap.push(temp); |
| maxHeap.push(minHeap.top()); |
| minHeap.pop(); |
|  |
| while(maxHeap.size()>minHeap.size()) { |
| minHeap.push(maxHeap.top()); |
| maxHeap.pop(); |
| } |
|  |
| vector<double>med; |
| if(minHeap.size() > maxHeap.size()) med.push\_back(minHeap.top()); |
| else if(maxHeap.size() == minHeap.size()) med.push\_back(double(maxHeap.top()+minHeap.top()) /2); |
|  |
| for(double i : med) cout << fixed << setprecision(1) << i << endl; |
| } |
| } |

|  |
| --- |
| **Prefix Sum.cpp** |
| #include<iostream> |
| #include<algorithm> |
| #include<vector> |
| using namespace std; |
|  |
| int prefixSum(vector<int> vec, int a, int b){ |
| vector<int> pSum (vec.size()); |
| pSum[0] = vec[0]; |
| for(int i=1;i<vec.size();i++){ |
| pSum[i] = pSum[i-1] + vec[i]; |
| } |
|  |
| int ret; |
| if(a>0) |
| ret = pSum[b] - pSum[a-1]; |
| else |
| ret = pSum[b]; |
|  |
| return ret; |
| } |
|  |
| int main(){ |
| vector<int> vec = {1,8,5,9,3,2,4,6}; |
| cout << prefixSum(vec, 3, 5) << endl; |
| } |

|  |
| --- |
| Maximum Element.cpp |
| #include<bits/stdc++.h> |
|  |
| #define inFile freopen("input.txt", "r", stdin); |
| #define sync ios\_base::sync\_with\_stdio(0);cin.tie(0);cout.tie(0); |
| using namespace std; |
|  |
| int main(){ |
| sync; |
| inFile; |
|  |
| int t; |
| cin >> t; |
| stack<long long> stk; |
| stack<long long> stkTemp; |
| vector<long long> vec; |
|  |
| while(t--){ |
| int query; |
| long long temp; |
| cin >> query; |
|  |
| switch(query){ |
| case 1: |
| cin >> temp; |
| //stk.push(temp); |
| vec.push\_back(temp); |
| break; |
| case 2: |
| //stk.pop(); |
| vec.pop\_back(); |
| break; |
| case 3: |
| long long maxx = \*max\_element(vec.begin(), vec.end()); |
| cout << maxx << endl; |
| break; |
| } |
|  |
| } |
| } |

|  |  |
| --- | --- |
| **BitWiseOperator.cpp** | **BitWiseOperatorTesting.cpp** |
| #include<iostream> | #include<iostream> |
| using namespace std; | using namespace std; |
|  |  |
| void calculate\_the\_maximum(int n, int k) { | int main(){ |
| int maxAnd = 0; | int a = 5; |
| int maxOr = 0; | int res = a>>1; |
| int maxXor = 0; | cout << res << endl; |
|  | // OUTPUT: 2 |
| for (int i=1; i<=n; i++) { |  |
| for (int j=i+1; j<=n; j++) { | res = a<<1; |
| int tempAnd = i&j; | cout << res << endl; |
| int tempOr = i|j; | // OUTPUT: 10 |
| int tempXor = i^j; |  |
|  | res = ~a; |
| if (tempAnd > maxAnd && tempAnd < k) | cout << res << endl; |
| maxAnd = tempAnd; | // OUTPUT: -6 (Homework) |
| if (tempOr > maxOr && tempOr < k) | } |
| maxOr = tempOr; |  |
| if (tempXor > maxXor && tempXor < k) |  |
| maxXor = tempXor; |  |
| } |  |
| } |  |
|  |  |
| cout << maxAnd << endl; |  |
| cout << maxOr << endl; |  |
| cout << maxXor << endl; |  |
| } |  |
|  |  |
| int main() { |  |
| int n, k; |  |
|  |  |
| cin >> n >> k; |  |
| calculate\_the\_maximum(n, k); |  |
| } |  |

|  |  |
| --- | --- |
| **Prefix Sum.cpp** | **Cumulative Sum Query.cpp** |
| #include<iostream> | int main(){ |
| #include<algorithm> | int n, sum=0; |
| #include<vector> | cin >> n; |
| using namespace std; | int array[n+10]; |
|  | int prefixSum[n+10]; |
| int prefixSum(vector<int> vec, int a, int b){ |  |
| vector<int> pSum (vec.size()); | for(int i=0; i<n; i++){ |
| pSum[0] = vec[0]; | cin >> array[i]; |
| for(int i=1;i<vec.size();i++){ | sum += array[i]; |
| pSum[i] = pSum[i-1] + vec[i]; | prefixSum[i] = sum; |
| } | } |
|  |  |
| int ret; | int q; |
| if(a>0) | cin >> q; |
| ret = pSum[b] - pSum[a-1]; | while(q--){ |
| else | int i, j; |
| ret = pSum[b]; | cin >> i >> j; |
|  | if(i==0) cout << prefixSum[j] << endl; |
| return ret; | else cout << prefixSum[j]-prefixSum[i-1] << endl; |
| } | } |
|  | } |
| int main(){ |  |
| vector<int> vec = {1,8,5,9,3,2,4,6}; |  |
| cout << prefixSum(vec, 3, 5) << endl; |  |
| } |  |

|  |  |
| --- | --- |
| **SET.cpp** | **Stack.cpp** |
| #include<iostream> | #include <bits/stdc++.h> |
| #include<set> |  |
| using namespace std; | using namespace std; |
|  |  |
| int main(){ | int main(){ |
| set<int> st; | cout << "Welcome to stack" << endl; |
| cout << "Is set empty? " << st.empty() << endl; |  |
|  | stack<int> stk; |
| st.insert(10); // O(logN) | stk.push(10); |
| st.insert(5); | stk.push(20); |
| st.insert(5); | stk.push(30); |
| st.insert(2); | stk.push(40); |
| st.insert(10); | stk.push(50); |
| st.insert(11); |  |
| st.insert(2); | while(!stk.empty()) { |
|  | cout << "Now on top: " << stk.top() << endl; |
| cout << "Is set empty? " << st.empty() << endl; | stk.pop(); |
|  | } |
|  | } |
| cout << "Traversing the set" << endl; |  |
| for(auto temp: st) cout << temp << endl; |  |
| } |  |

|  |  |
| --- | --- |
| **MAP-2.cpp** | **MAP.cpp** |
| #include<iostream> | #include<bits\stdc++.h> |
| #include<map> | using namespace std; |
| #include<vector> |  |
| #include<algorithm> | int main(){ |
| using namespace std; | map<int, int> mp; |
|  | set<int> st; |
| /\*struct students{ |  |
| int id; | mp[0] = 1; |
| string name; | mp[1] = 2; |
| }st[100]; | mp[2] = 3; |
|  |  |
| for(students temp :st){ | st.insert(10); |
|  | st.insert(1); |
| }\*/ | st.insert(5); |
|  | st.insert(3); |
| /\*bool cmp(pair<char, int>& a, pair<char, int>& b){ |  |
| return a.second < b.second; | for(auto t: mp) cout << "index: " << t.first << " value: " << t.second << endl; |
| }\*/ |  |
|  | for(auto t: st) cout << t << endl; |
| int main(){ | } |
|  |  |
| map<char, int> mp; |  |
| mp['z'] = 100; // O(logn) |  |
| mp['a'] = 200; |  |
| mp['m'] = 50; |  |
| mp['c'] = 70; |  |
|  |  |
| //cout << mp['a'] << endl; |  |
|  |  |
| /\*for(pair<char, int> pr: mp){ |  |
| cout << pr.first << " -> " << pr.second << endl; |  |
| }\*/ |  |
|  |  |
| for(auto pr: mp){ |  |
| cout << pr.first << " -> " << pr.second << endl; |  |
| } |  |
|  |  |
| cout << endl << endl; |  |
| cout << "Pair of vector" << endl; |  |
|  |  |
| vector<pair<int, int>> vec; |  |
| vec.push\_back(make\_pair(1,100)); |  |
| vec.push\_back(make\_pair(2,150)); |  |
| vec.push\_back(make\_pair(3,70)); |  |
| vec.push\_back(make\_pair(4,60)); |  |
| vec.push\_back(make\_pair(5,90)); |  |
|  |  |
| /\*for(pair<int, int> pr: vec){ |  |
| cout << pr.first << " -> " << pr.second << endl; |  |
| }\*/ |  |
|  |  |
| for(auto pr: vec) cout << pr.first << " -> " << pr.second << endl; |  |
| } |  |