**Sublime Build : Windows**

{

"cmd": ["g++.exe","-std=c++20", "${file}", "-o", "${file\_base\_name}.exe", "&&" , "${file\_base\_name}.exe<inputf.in>outputf.in"],

"selector":"source.cpp",

"shell":true,

"working\_dir":"$file\_path"

}

**Sublime Build : Linux**

{

"cmd" : ["g++ -std=c++20 $file\_name -o $file\_base\_name && timeout 10s ./$file\_base\_name<inputf.in>outputf.in"],

"selector" : "source.cpp",

"shell": true,

"working\_dir" : "$file\_path"

}

**Fast Input/Output**

ios\_base::sync\_with\_stdio(false); cin.tie(NULL); cout.tie(NULL);

#pragma GCC optimize("O3,unroll-loops")

#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")

**Priority Queue**

priority\_queue<int, vector<int>,greater<int>> pq;

// min value comes first

**PBDS (Ordered Set)**

#include <ext/pb\_ds/assoc\_container.hpp>

#include <ext/pb\_ds/tree\_policy.hpp>

using namespace \_\_gnu\_pbds;

template<class T> using ordered\_set = tree<T, null\_type, less<T>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;

**// s.order\_of\_key(x) = number of elements strictly less than x**

**// \*s.find\_by\_order(i) = ith element in set (0 index)**

**GpHashTable**

#include <ext/pb\_ds/assoc\_container.hpp>

using namespace \_\_gnu\_pbds;

struct customHash {

static uint64\_t saimur(uint64\_t x) {

x += 0x9e3779b97f4a7c15;

x = (x ^ (x >> 30)) \* 0xbf58476d1ce4e5b9;

x = (x ^ (x >> 27)) \* 0x94d049bb133111eb;

return x ^ (x >> 31);

}

size\_t operator()(uint64\_t x) const {

static const uint64\_t FIXED\_RANDOM = chrono::steady\_clock::now().time\_since\_epoch().count();

return saimur(x + FIXED\_RANDOM);

}

};

gp\_hash\_table <int, int, customHash> table;

**Bit Manipulation**

#define setbit(n) \_\_builtin\_popcountll(n) // Count of set bit

#define zerobit(n) \_\_builtin\_ctzll(n) // Count of 0 bit before 1st 1

#define getbit(n,i) ((n & (1LL << i)) != 0) // nth bit ?

#define clearlsb(n,i) (n & (~((1<<(i+1))-1))) // unset upto ith bit rtl

#define clearmsb(n,i) (n & ((1<<(i+1))-1)) // unset before ith bit ltr

#define setbit0(n,i) (n & (~(1LL << i))) // unset any bit

#define setbit1(n,i) (n | (1LL << i)) // set any bit

#define togglebit(n,i) (n ^ (1LL << i))

**Important Tips**

Modular Inverse : **binExp(a,M-2,M)**

50^(64^32) : **binExp(50,binExp(64,32,M-1),M)**

Upper Diagonal of a ChessBoard : **(n-1)+(col-row)**

Lower Diagonal of a ChessBord : **(row +col)**

**=========== Algorithm ============**

**Seive**

const int N = 1e7 + 10;

vector<bool> isPrime(N,1);

vector<int> lowestPrime(N), highestPrime(N);

void seive()

{

isPrime[0] = isPrime[1] = false;

for(int i=2; i<N; i++)

{

if(isPrime[i])

{

lowestPrime[i] = highestPrime[i] = i;

for(int j = 2 \* i; j<N; j+=i)

{

isPrime[j] = false; highestPrime[j] = i;

if(lowestPrime[j]==0) lowestPrime[j] = i;

}

}

}

}

**Stored Prime Seive**

const int N = 1e7 + 10;

vector<bool> isPrime(N, 1);

vector<int> prime;

void seive()

{

isPrime[0] = isPrime[1] = false;

for (int i = 2; i < N; i++)

{

if (isPrime[i])

{

for (int j = 2 \* i; j < N; j += i)

{

isPrime[j] = false;

}

}

}

prime.push\_back(0);

for (int i = 2; i < N; i++) {

if (isPrime[i]) {

prime.push\_back(i);

}

}

}

**Phi Using Seive Variation in Range 1 to N**

void phi\_1\_to\_n(int n) { **//nlogn**

vector<int> phi(n + 1);

phi[0] = 0; phi[1] = 1;

for (int i = 2; i <= n; i++)

phi[i] = i - 1;

for (int i = 2; i <= n; i++)

for (int j = 2 \* i; j <= n; j += i)

phi[j] -= phi[i];

}

int phi(int n) { **//sqrt(n)**

int result = n;

for (int i = 2; i \* i <= n; i++) {

if (n % i == 0) {

while (n % i == 0) n /= i;

result -= result / i;

}

}

if (n > 1) result -= result / n;

return result;

}

**Phi for Single Number**

int phi(int n) {

int result = n;

for (int i = 2; i \* i <= n; i++) {

if (n % i == 0) {

while (n % i == 0) n /= i;

result -= result / i;

}

}

if (n > 1) result -= result / n;

return result;

}

**Prime Factorization Using Seive**

map<int,int> prime\_factors; // first = prime factors & second = count

while(num>1){

int prime\_factor = lowestPrime[num];

while(num % prime\_factor == 0){

num /= prime\_factor;

prime\_factors[prime\_factor]++;

}

}

**Divisors**

const int N = 1e5+10;

vector<int> divisors[N];

void divisorSeive(){

for(int i = 1; i < N; i++){

for(int j = i; j < N; j += i){

divisors[j].push\_back(i);

}

}

}

// P^x \* Q^y \* R^z => here, P, Q, R are prime factors & x, y, z are powers

NOD = (x + 1) (y + 1) (z + 1)

**Number of Divisors (NOD)**

int NOD(int n) {

int nod = 1;

for (int i = 2; i \* i <= n; ++i) {

int pow = 0;

while (n % i == 0) {

pow++;

n /= i;

}

nod \*= (pow + 1);

}

if (n > 1) nod \*= 2;

return nod;

}

SOD = ((P^(x+1)-1)/(P-1)) \* ((Q^(y+1)-1)/(Q-1)) \* ((R^(z+1)-1)/(R-1))

**Sums of Divisors (SOD)**

int SOD(int n) {

int sod = 1;

for (int i = 2; i \* i <= n; ++i) {

if (n % i == 0) {

int pow = 1;

while (n % i == 0) {

pow \*= i; // p^e

n /= i;

}

pow \*= i; // p^e+1

sod \*= (pow - 1) / (i - 1); //(p^e+1)-1 / p-1

}

}

if (n > 1) sod \*= (n + 1);

return sod;

}

**Miller-Rabin (Probable Primality Check)**

**// import BigMod**

bool miller\_rabin(int n) {

if (n <= 1 || n % 2 == 0) {

if (n != 2) {

return false;

}

}

if (n == 2 || n == 3) {

return true;

}

int d = n - 1;

while (d % 2 == 0) {

d /= 2;

}

int a[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};

for (int i = 0; i < 12 && a[i] < n; ++i) {

int temp = d;

int mod = binExp(a[i], temp, n);

if (mod == 1) {

continue;

}

while (temp != n - 1 && mod != n - 1) {

mod = binMultiply(mod, mod, n);

temp \*= 2;

}

if (mod != n - 1) {

return false;

}

}

return true;

}

**Base Conversion Any to Any**

int any\_to\_anybase(int num, int baseA, int baseB) {

int ans = 0;

int dec = stoll(to\_string(num), nullptr , baseA);

int rem = 0;

int i = 1;

while (dec) {

rem = dec % baseB;

dec /= baseB;

ans += rem \* i;

i \*= 10;

}

return ans;

}

**Length of LIS**

int LIS\_Length() {

ans.push\_back(v[0]);

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

if (v[i] > ans.back()) ans.push\_back(v[i]);

int x = lower\_bound(ans.begin(), ans.end(), v[i]) - ans.begin();

ans[x] = v[i];

}

return ans.size();

}

**Print LIS (Binary Search)**

int n; cin >> n;

int a[n], parent[n], ind[n];

vector<int> lis;

memset(parent, -1, sizeof parent);

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

cin >> a[i];

int it = lower\_bound(lis.begin(), lis.end(), a[i]) - lis.begin();

if (it == lis.size()) {

lis.push\_back(a[i]);

ind[lis.size() - 1] = i;

parent[i] = (lis.size() == 1 ? -1 : ind[it - 1]);

}

else {

lis[it] = a[i];

ind[it] = i;

parent[i] = (it == 0 ? -1 : ind[it - 1]);

}

}

vector<int>LIS;

int it = ind[lis.size() - 1];

LIS.push\_back(lis.back());

while (parent[it] != -1) {

it = parent[it];

LIS.push\_back(a[it]);

}

**Maximum Subarray Sum (Kadane’s Algo)**

int max\_subarray\_sum(deque<int> &a) {

int ans = -1e9, sum = 0;

for (int i = 0; i < a.size(); i++) {

sum = max(a[i], sum + a[i]);

ans = max(ans, sum);

}

return ans;

}

**Maximum Subarray size thats sum = K**

int n, k; cin >> n >> k;

int total\_sum = 0;

vector < int > pre(n + 7, 0);

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

int temp; cin >> temp;

total\_sum += temp;

if (i == 1) pre[i] = temp;

else pre[i] = pre[i - 1] + temp;

}

if (total\_sum < k) { cout << "-1" << endl; return; }

if (total\_sum == k) { cout << "0" << endl; return; }

int maximum\_subSize = 0;

gp\_hash\_table < int, int, customHash> table;

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

if (pre[i] >= k) {

int subSUM = pre[i] - k;

if (subSUM == 0) maximum\_subSize = max(maximum\_subSize, i);

else if (table[subSUM]) {

int left = table[subSUM];

int right = i;

int subSize = right - left;

maximum\_subSize = max(subSize, maximum\_subSize);

}

}

if (!table[pre[i]]) table[pre[i]] = i;

}

cout << maximum\_subSize << endl;

**Number of Subarray Sum = K**

int n, k; cin >> n >> k;

int total\_sum = 0;

vector < int > pre(n + 7, 0);

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

int temp; cin >> temp;

total\_sum += temp;

if (i == 1) pre[i] = temp;

else pre[i] = pre[i - 1] + temp;

}

int cnt\_subarry = 0;

gp\_hash\_table < int, int, customHash> table;

table[0] = 1;

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

cnt\_subarry += table[pre[i] - k];

table[pre[i]]++;

}

cout << cnt\_subarry << endl;

**Geometric Sum**

int MOD = 1e9 + 7;

int BigMod(int b, int p) {

int x = 1;

while (p) {

if (p & 1) x = (x \* b) % MOD;

b = (b \* b) % MOD;

p >>= 1;

} return x;

}

int geometric\_Sum() {

int a, r, n; cin >> a >> r >> n; //a = first value r = ratio, n = n-term

int res = BigMod(r, n);

int numerator = (a \* (1 - res)) % MOD;

numerator = (numerator + MOD) % MOD;

int denominator = ((1 - r) % MOD + MOD) % MOD;

int ans = (numerator \* BigMod(denominator, MOD - 2)) % MOD;

return ans;

}

**GCD LCM Notes**

gcd(a,gcd(b,c))=gcd(gcd(a,b),c)

gcd(a,b,c)=gcd(gcd(a,b),c)

gcd(a,b)=gcd(a−b,b)

lcm(a, gcd(b,c)) = gcd(lcm(a, b), lcm(a, c))

gcd(a, lcm(b, c)) = lcm(gcd(a, b), gcd(a, c))

**Subset Generation using bitmask**

vector<vector<int>> subsets(vector<int> &v){

int n = v.size();

int subset\_cnt = (1<<n);

vector<vector<int>> subsets;

for(int i=0; i<subset\_cnt; i++){ // Here, i for mask value

vector<int> subset;

for(int j = 0; j<n; j++)

if(i & (1<<j)) subset.pb(v[j]);

subsets.pb(subset);

} return subsets;

}

**Extended Euclidean Algorithm (GCD)**

int egcd(int a, int b, int &x, int &y) {

if (b == 0) {

x = 1;

y = 0;

return a;

}

int x1, y1;

int d = egcd(b, a % b, x1, y1);

x = y1; y = x1 - y1 \* (a / b);

return d;

}

**Merge Sort**

void merge(int l, int r, int mid) {

int l\_sz = mid - l + 1;

int L[l\_sz + 1];

int r\_sz = r - mid;

int R[r\_sz + 1];

for(int i=0; i<l\_sz; i++) {

L[i] = arr[i+l];

}

for(int i=0; i<r\_sz; i++) {

R[i] = arr[i+mid+1];

}

L[l\_sz] = R[r\_sz] = INT\_MAX;

int l\_i = 0;

int r\_i = 0;

for(int i=l; i<=r; i++) {

if(L[l\_i] <= R[r\_i]) {

arr[i] = L[l\_i];

l\_i++;

}

else {

arr[i] = R[r\_i];

r\_i++;

}

}

}

void mergeSort(int l, int r) {

if(l == r) return;

int mid = (l+r)/2;

mergeSort(l,mid);

mergeSort(mid+1,r);

merge(l,r,mid);

}

**Next Greater Element**

vector<int> NGE(vector<int> &v){

vector<int> nge(v.size());

stack<int> st;

for(int i=0; i<v.size(); i++){

while(!st.empty() && v[i]>v[st.top()]){

nge[st.top()] = i;

st.pop();

}

st.push(i);

}

while(!st.empty()){

nge[st.top()] = -1;

st.pop();

}

return nge;

}

**vector<int> nge = NGE(v); // Main Function calling**

**n-th root of any number (binary search)**

double lo = 1, hi = x, mid;

while(hi - lo > eps){ // eps = 1e-6, if want 5 digit accurate

mid = (hi+lo)/2;

if(multiply(mid,3) < x) lo = mid;

else hi = mid;

}

**binExp(b,M-2,M) calculates inverse of b**

(a / b) % M = a \* binExp(b, M - 2, M)

**Binary Exponentiation**

int binMultiply(int a, int b, int M){

int res = 0;

while(b){

if(b & 1) res = (res + a) % M;

a = (a + a) % M;

b >>= 1;

}

return res;

}

int binExp(int a, int b, int M){

int res = 1;

while(b){

if(b & 1) res = binMultiply(res,a,M);

a = binMultiply(a,a,M);

b >>= 1;

}

return res;

}

**binExp(50,binExp(64,32,mod-1),mod); // function calling**

// phi(M) = M \* multiply\_of(1-1/P) here, P is all distinct prime number

// (a^b)%M = (a^(b%(M-1)))%M // ETF --> Euler Totient Function

**n-th Fibonacci Number using Matrix Exponentiation**

const int mod = 1e9 + 7;

const int sz = 2;

struct Mat {

int mat[sz][sz];

Mat() {

memset(mat, 0, sizeof(mat));

}

void identity() {

for (int i = 0; i < sz; i++) {

mat[i][i] = 1;

}

}

Mat operator\* (Mat a) {

Mat res;

for (int i = 0; i < sz; i++) {

for (int j = 0; j < sz; j++) {

for (int k = 0; k < sz; k++) {

res.mat[i][j] += mat[i][k] \* a.mat[k][j];

res.mat[i][j] %= mod;

}

}

}

return res;

}

};

int Fib(int n) {

Mat res;

res.identity();

Mat T;

T.mat[0][0] = T.mat[0][1] = T.mat[1][0] = 1;

if (n <= 2) return 1;

n -= 2;

while (n) {

if (n & 1) res = res \* T;

T = T \* T;

n /= 2;

}

return (res.mat[0][0] + res.mat[0][1]) % mod;

}

**n-th Permutation**

template <typename T>

vector<T> nthPermutation(vector<T>&v, int k) {

vector<T>nums(v.begin(), v.end());

int n = nums.size();

vector<T> result;

int64\_t fact = 1;

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

fact \*= i;

}

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

fact /= (n - i);

int idx = (k - 1) / fact;

result.push\_back(nums[idx]);

nums.erase(nums.begin() + idx);

k -= idx \* fact;

}

return result;

}

**int128 read and write**

\_\_int128 read() {

\_\_int128 x = 0, f = 1;

char ch = getchar();

while (ch < '0' || ch > '9') {

if (ch == '-') f = -1;

ch = getchar();

}

while (ch >= '0' && ch <= '9') {

x = x \* 10 + ch - '0';

ch = getchar();

}

return x \* f;

}

void print(\_\_int128 x) {

if (x < 0) {

putchar('-');

x = -x;

}

if (x > 9) print(x / 10);

putchar(x % 10 + '0');

}

**Large Number MOD**

bool largeNumberMod(string number,int M) // largeNumberMod("12345",10);

{

int now = number[0] - '0';

for(int i = 1; i < number.length(); i++) {

now = ((now\*10) + number[i]-'0') % M;

}

**// if now == 0 then it is possible to divide the number by M**

if(now == 0) return true;

else return false;

}

**Two Large Number Multiplication**

string toString(int num)

{

if (num == 0) return "0";

string s = "";

while (num)

{

s += (num % 10) + '0';

num /= 10;

}

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

return s;

}

string multiply(string num1, string num2) //string ans = multiply("123","125");

{

int n1\_sz = num1.size();

int n2\_sz = num2.size();

if (num1 == "0" || num2 == "0") return "0";

vector<int> result(n1\_sz + n2\_sz, 0);

int i\_n1 = 0;

int i\_n2 = 0;

for (int i = n1\_sz - 1; i >= 0; i--) {

int carry = 0;

int n1 = num1[i] - '0';

i\_n2 = 0;

for (int j = n2\_sz - 1; j >= 0; j--) {

int n2 = num2[j] - '0';

int sum = n1 \* n2 + result[i\_n1 + i\_n2] + carry;

carry = sum / 10;

result[i\_n1 + i\_n2] = sum % 10;

i\_n2++;

}

if (carry > 0) result[i\_n1 + i\_n2] += carry;

i\_n1++;

}

int i = result.size() - 1;

while (i >= 0 && result[i] == 0) i--;

if (i == -1) return "0";

string s = "";

while (i >= 0)

s += toString(result[i--]);

return s;

}

**nCr**

const int M = 1e9 + 7;

const int MAXN = 1e6 + 5;

int fact[MAXN], invFact[MAXN];

int binExp(int a, int b) {

int res = 1;

while (b) {

if (b & 1) res = (res \* a) % M;

a = (a \* a) % M;

b >>= 1;

}

return res;

}

int inverse(int x) {

return binExp(x, M - 2);

}

void compute\_factorial() {

fact[0] = invFact[0] = 1;

for (int i = 1; i < MAXN; i++) {

fact[i] = (fact[i - 1] \* i) % M;

invFact[i] = inverse(fact[i]);

}

}

int nCr(int N, int R) {

if (N < 0 || R < 0 || R > N) {

return 0;

}

int num = fact[N];

int denom = (invFact[R] \* invFact[N - R]) % M;

return (num \* denom) % M;

}

**============== Graph ==============**

**DFS**

const int N = 1e5 + 5;

vector<int> g[N];

vector<bool>visited(N);

void dfs(int index) {

visited[index] = true;

for (auto &child : g[index]) {

if (!visited[child]) {

dfs(child);

}

}

}

**LCA**

const int N = 1e5 + 5;

vector<int> g[N], parent(N), depth(N, 0);

void dfs(int vertex, int par = -1) {

parent[vertex] = par;

for (auto child : g[vertex]) {

if (child != par) {

depth[child] = depth[vertex] + 1;

dfs(child, vertex);

}

}

}

int lca(int x, int y) {

int diff = min(depth[x], depth[y]);

while (depth[x] > diff) x = parent[x];

while (depth[y] > diff) y = parent[y];

while (x != y) {

x = parent[x];

y = parent[y];

}

return x;

}  
  
**Graph Path**

vector<int> path(int vertex) {

vector<int>ans;

while (vertex != -1) {

ans.push\_back(vertex);

vertex = parent[vertex];

}

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

return ans;

}

**0/1 BFS**

const int N = 1e5 + 5, INF = 1e9;

vector<pair<int, int>> g[N];

vector<int> level(N, INF);

int n, m;

int bfs() {

deque<int> dq;

dq.push\_back(1);

level[1] = 0;

while (dq.size()) {

int vertex = dq.front();

dq.pop\_front();

for (auto [child, wt] : g[vertex]) {

if (level[vertex] + wt < level[child]) {

level[child] = level[vertex] + wt;

if (wt == 1) {

dq.push\_back(child);

}

else {

dq.push\_front(child);

}

}

}

}

return level[m] == INF ? -1 : level[m];

}

**Graph Path**

const int N = 1e5 + 5, INF = 1e18 + 7;

vector<pair<int, int>> g[N];

bool visited[N];

vector<int> dist(N, INF), parent(N);

bool dijkstra(int source) {

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;

pq.push({0, source});

dist[source] = 0;

parent[source] = -1;

while (pq.size()) {

int x = pq.top().second;

pq.pop();

if (visited[x]) continue;

visited[x] = 1;

for (auto [child\_x, child\_wt] : g[x]) {

if (dist[x] + child\_wt < dist[child\_x]) {

parent[child\_x] = x;

dist[child\_x] = child\_wt + dist[x];

pq.push({dist[child\_x], child\_x});

}

}

}

return (dist[n] == INF);

}

**// sort pair by 2nd value => sorta -> ascending & sortd-> decending**

bool sorta(const pair<int, int> &a, const pair<int, int> &b) {return (a.second < b.second);}

bool sortd(const pair<int, int> &a, const pair<int, int> &b) {return (a.second > b.second);}

// Implementation : sort(v.begin(),v.end(),sorta); Here, v is vector of pairs

**Segment Tree (Max Subarray Sum + Point Update)**

struct var {

int max\_sum, pref, suf, sum;

};

struct SGTree {

vector<var> seg;

int n;

SGTree(int size) {

n = size;

seg.resize(4 \* n + 1);

}

var NUTRAL\_ELEMENT = {0,0,0,0};

var single(int x) {

if (x > 0) return {x, x, x, x};

else return {0, 0, 0, x};

}

var calc(var left, var right) {

return {

max({left.max\_sum, right.max\_sum, left.suf + right.pref}),

max(left.pref, left.sum + right.pref),

max(right.suf, left.suf + right.sum),

left.sum + right.sum

};

}

void build(int index, int low, int high, int a[]) {

if (low == high) {

seg[index] = single(a[low]);

return;

}

int mid = (low + high) / 2;

build(2 \* index, low, mid, a);

build(2 \* index + 1, mid + 1, high, a);

seg[index] = calc(seg[2 \* index], seg[2 \* index + 1]);

}

void build(int a[]) {

build(1, 1, n, a);

}

var query(int index, int low, int high, int l, int r) {

if (low >= l && high <= r) return seg[index];

if (low > r || high < l) return NUTRAL\_ELEMENT;

int mid = (low + high) / 2;

var left = query(2 \* index, low, mid, l, r);

var right = query(2 \* index + 1, mid + 1, high, l, r);

return calc(left, right);

}

var query(int l, int r) {

return query(1, 1, n, l, r);

}

void update(int index, int low, int high, int i, int x) {

if (low == high) {

seg[index] = single(x);

return;

}

int mid = (low + high) / 2;

if (i <= mid) update(2 \* index, low, mid, i, x);

else update(2 \* index + 1, mid + 1, high, i, x);

seg[index] = calc(seg[2 \* index], seg[2 \* index + 1]);

}

void update(int index, int value) {

update(1, 1, n, index, value);

}

};

**SegTree Lazy Propagation (Range Update, Query)**

struct node {

int value = 0;

int lazy = 0;

};

struct SGTree {

vector<node> seg;

int n;

SGTree(int size) {

n = size;

seg.resize(4 \* n + 1);

}

int NUTRAL\_ELEMENT = 0;

void build(int index, int low, int high, int a[]) {

if (low == high) {

seg[index].value = a[low];

return;

}

int mid = (low + high) / 2;

build(2 \* index, low, mid, a);

build(2 \* index + 1, mid + 1, high, a);

seg[index].value = seg[2 \* index].value + seg[2 \* index + 1].value;

}

void build(int a[]) {

build(1, 1, n, a);

}

int query(int index, int low, int high, int l, int r, int carry) {

if (low > r || high < l) return NUTRAL\_ELEMENT;

if (low >= l && high <= r) return seg[index].value + carry \* (high - low + 1);

int mid = (low + high) / 2;

carry += seg[index].lazy;

int left = query(2 \* index, low, mid, l, r, carry);

int right = query(2 \* index + 1, mid + 1, high, l, r, carry);

return left + right;

}

int query(int l, int r) {

return query(1, 1, n, l, r, 0);

}

void update(int index, int low, int high, int l, int r, int x) {

if (low > r || high < l) return;

if (low >= l && high <= r) {

seg[index].value += (high - low + 1) \* x;

seg[index].lazy += x;

return;

}

int mid = (low + high) / 2;

update(2 \* index, low, mid, l, r, x);

update(2 \* index + 1, mid + 1, high, l, r, x);

seg[index].value = seg[2 \* index].value + seg[2 \* index + 1].value + (high - low + 1) \* seg[index].lazy;

}

void update(int left, int right, int value) {

update(1, 1, n, left, right, value);

}

};

**Hungarian Algorithm (Assignment Problem)**

const int INF = INT\_MAX;

int n;

vector<int> u, v, p, way;

int hungarian(vector<int>& assignment, vector<vector<int>>& matrix) {

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

p[0] = i;

int j0 = 0;

vector<int> minv(n + 1, INF);

vector<bool> used(n + 1, false);

do {

used[j0] = true;

int i0 = p[j0], delta = INF, j1;

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

if (!used[j]) {

int cur = matrix[i0 - 1][j - 1] - u[i0] - v[j];

if (cur < minv[j]) {

minv[j] = cur;

way[j] = j0;

}

if (minv[j] < delta) {

delta = minv[j];

j1 = j;

}

}

}

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

if (used[j]) {

u[p[j]] += delta;

v[j] -= delta;

} else {

minv[j] -= delta;

}

}

j0 = j1;

} while (p[j0] != 0);

do {

int j1 = way[j0];

p[j0] = p[j1];

j0 = j1;

} while (j0 != 0);

}

int totalCost = -v[0];

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

assignment[p[j] - 1] = j - 1;

}

return totalCost;

}

int main() {

vector<vector<int>> matrix;

cin >> n;

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

vector<int> w(n);

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

cin >> w[j];

}

matrix.push\_back(w);

}

u.resize(n + 1);

v.resize(n + 1);

p.resize(n + 1);

way.resize(n + 1);

vector<int> assignment(n, -1);

int result = hungarian(assignment, matrix);

cout << result << '\n';

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

cout << i + 1 << " " << assignment[i] + 1 << endl;

}

return 0;

}

**============= Math =============**

**Distance Between Two Poins**

double point\_dist(int a, int b, int x, int y) {

return sqrt(((x - a) \* (x - a)) + ((y - b) \* (y - b)));

}

bool isPowerOfTwo(int n) {return !(n & (n - 1));}

bool isPerfectSquare(int x) {if (x >= 0) {int sr = sqrt(x); return (sr \* sr == x);} return false;}

**Equation of a line from two point** ax+by=c , where

a = y2 - y1; b = x1 - x2; c = ax1 + by1

**Check if the line intersects a circle** if(ans <= radius)

(abs(a\*x + b\*y + c)/sqrt(a\*a+b\*b))

**Area of a regular hexagon** A = (3\*sqrt(3)\*a\*a)/2;

**Number of intersecting points** made by the diagonals of a n-gon : **(n\*(n-1)\*(n-2)\*(n-3))/24 or nC4**

**Number of Diagonals of any n-gon : nC2-n**

**Another Equation : (n \* (n-3))/2**

**nCr = n-1Cr-1 + n-1Cr**

**Multiplication of first n odd numbers : 2n! / (2**n\* **n!)**

**Summation of first n odd numbers : n2**

**Summation of first n even numbers : n(n+1)**

**============ Debug =============**

#define debug(x) cerr << #x <<" "; \_print(x); cerr << endl;

void \_print(int t) {cerr << t;}

void \_print(string t) {cerr << t;}

void \_print(char t) {cerr << t;}

void \_print(double t) {cerr << t;}

template <class T, class V> void \_print(pair <T, V> p);

template <class T> void \_print(vector <T> v);

template <class T> void \_print(set <T> v);

template <class T, class V> void \_print(map <T, V> v);

template <class T> void \_print(multiset <T> v);

template <class T, class V> void \_print(pair <T, V> p) {cerr << "{"; \_print(p.first); cerr << ","; \_print(p.second); cerr << "}";}

template <class T> void \_print(vector <T> v) {cerr << "[ "; for (T i : v) {\_print(i); cerr << " ";} cerr << "]";}

template <class T> void \_print(set <T> v) {cerr << "[ "; for (T i : v) {\_print(i); cerr << " ";} cerr << "]";}

template <class T> void \_print(multiset <T> v) {cerr << "[ "; for (T i : v) {\_print(i); cerr << " ";} cerr << "]";}

template <class T, class V> void \_print(map <T, V> v) {cerr << "[ "; for (auto i : v) {\_print(i); cerr << " ";} cerr << "]";}

**gen.cpp**

int rand(int a, int b){

return a + rand() % (b - a + 1);

}

int main(int argc, char\* argv[]){

srand(atoi(argv[1]));

// gen code here, run by ./gen (n)

}

**s.sh**

for((i = 1; ; i++)); do

echo $i

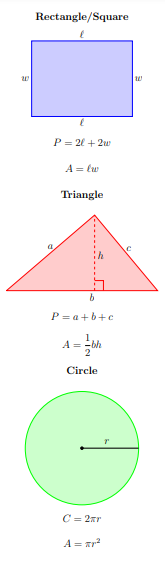
./gen $i > inputf.in

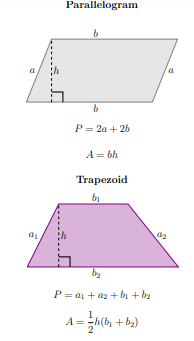
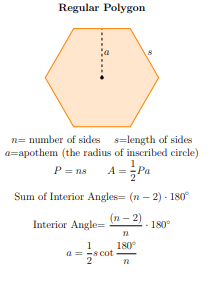
./main < inputf.in > outputf.in

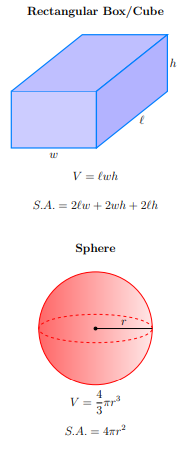
./brute < inputf.in > output2.in

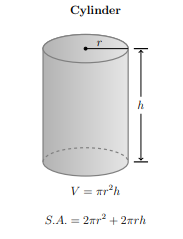
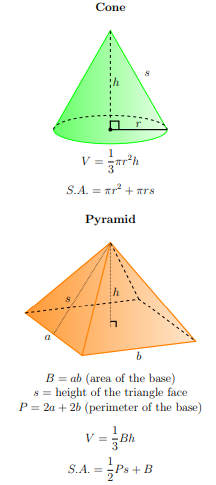
diff -w outputf.in output2.in || break

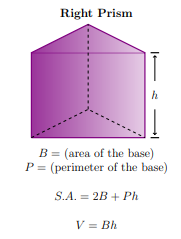
done

**Geometry**









***Number Series***

1. nc1 + nc2 + nc3 + .... + nCn = (2^n) - 1

2. 1^2 + 2^2 + 3^2 + 4^2 + ...... + n^2 = (n \* (n + 1)(2 \* n + 1)) / 6

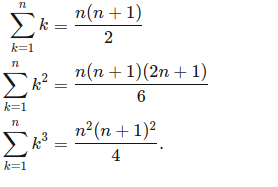
3. 2^0 + 2^1 + 2^3 + ....... + 2^n = (2^(n+1)) - 1

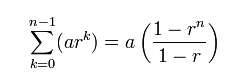
4. nCr = (nC(r-1)) \* ((n-r+1)/r)

5. (k^0 + k^1 + ……+k^n) < k^(n+1) true for all k>=2

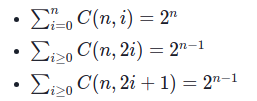
**5.2** (1 + a + a^2 + a^3 ....... a^(n-1)) = (a^n - 1)/(a-1)

6. Sum of first n positive odd integers is n^2

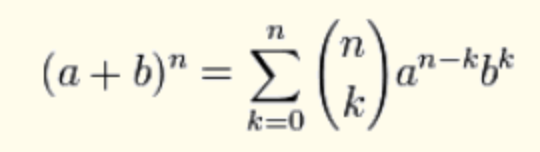
7.   
  
  
  
  
  
  
  
  
  
 8. a + (a+d) + (a+2d) + (a+3d) + …

9. a + ar + ar^2 + ... + ar^(n-1)

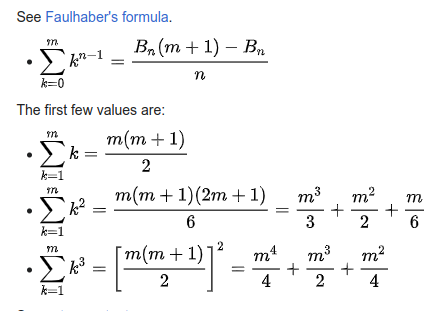
10. C(n,k)=C(n,n−k)

11. C(n,k)=C(n−1,k−1)+C(n−1,k)

12.



13.   
  
  
14. 



15.   
  
  
  
  
  
  
  
  
  
  
16.   
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
