|  |  |
| --- | --- |
| //Binary Search  //FFFFFFFFFTTTTTTTTTT | //Binary Search  //TTTTTTTTTTFFFFFFFFFFF |
| ll st = 0, en = 1e18;  ll mid;  while (st < en) {  mid = st + (en - st) / 2;  if (!valid(mid))  st = mid + 1;  else  en = mid;  } | ll st = 0, en = 1e18;  ll mid;  while (st < en) {  mid = st + (en - st + 1) / 2;  if (valid(mid))  st = mid;  else  en = mid - 1;  } |
| //Binary Search (double) : | double st = 0, en = 1e9;  double size = en - st;  for (size /= 2; size > 1e-7; size /= 2) {  if (!valid(st + size))  st += size;  } |

|  |
| --- |
| //Graph setup (adjacency list) |
| const int N = 1e3 + 5, M = N \* N + 5, OO = 0x3f3f3f3f;  int head[N], nxt[M], to[M], ne;  void add\_edge(int f, int t) {  nxt[ne] = head[f];  to[ne] = t;  head[f] = ne++;  }  void add\_biedge(int a, int b) { add\_edge(a, b), add\_edge(b, a); }  #define neigh(u,e,v) for(int e = head[u], v ; v = to[e], ~e ; e = nxt[e]) |
| //Graph setup (DSU) |
| int sz[N], par[N], sets;  void init(int n) {  iota(par, par + n, 0);  fill(sz, sz + n, 1);  sets = n;  }  int find(int u) {  return (par[u] == u ? u : par[u] = find(par[u]));  }  void join(int a, int b) {  a = find(a), b = find(b);  if (a == b) return;  if (sz[a] < sz[b]) swap(a, b); //optional  sz[a] += sz[b]; //remove if no size  par[b] = a;  --sets;  } |

|  |
| --- |
| //Topology BFS  in[t]++; //in add\_edge |
| struct cmp{  bool operator () (const int & a, const int & b) const{  return a>b;  }  };  void BFS(){  priority\_queue<int, vector<int>, cmp> q;  for(int i = 1 ; i <= n ; ++i)  if(!in[i]) q.push(i);  while(!q.empty()){  int u = q.top();  result.push\_back(u);  q.pop();  neigh(u, e, v) {  if (!--in[v])  q.push(v);  }  }  } |

|  |
| --- |
| void Floyd(){  vector<vector<int> > & ret = adjMat;  for(int k = 0 ; k < N ; ++k)  for(int i = 0 ; i < N ; ++i)  for(int j = 0 ; j < N ; ++j)  ret[i][j] = min(ret[i][j], ret[i][k] + ret[k][j]);  } |
| void Dijkstra(int src){  memset(dis, OO, sizeof dis);  priority\_queue<pair<int, int> > q;  q.push({0, src});  dis[src] = 0;  while(q.size()){  int d = -q.top().first, u = q.top().second;  q.pop();  if(dis[u] != d) continue;  for(int k = head[u] ; ~k ; k = nxt[k]){  int v = to[k], c = cst[k];  int dd = d+c;  if(dis[v] > dd){  dis[v] = dd;  q.push({-dd, v});  }  }  }  } |
| void DijkstraN2(int src){  memset(dis, OO, sizeof dis); //Infinity  iota(notVis, notVis+n, 0);  dis[src] = 0;  int sz = n, nxt = src;  while (nxt != -1){  int i = notVis[nxt];    swap(notVis[nxt], notVis[--sz]); //Mark as visited    int lowest = OO;  nxt = -1;  for(int k = 0 ; k < sz ; ++k){ //For all non-visited nodes  int j = notVis[k];  if(dis[j] > dis[i] + adjMat[i][j]){  dis[j] = dis[i] + adjMat[i][j];  }  if(dis[j] < lowest){  lowest = dis[j];  nxt = k;  }  }  }  } |

|  |
| --- |
| //Strings |
| #include <bits/stdc++.h>  using namespace std;  const int N = 1e5 + 1;  char pat[N], str[N];  int F[N];  //I need to know what is the maximum number of characters that can be matched if c is included  int getNewLen(int len, char c) {  while (len && c != pat[len])  len = F[len - 1];  return len + (c == pat[len]);  }  //00012345  //abcabcab  void computeF() {  F[0] = 0;  for (int i = 1; pat[i]; i++)  F[i] = getNewLen(F[i - 1], pat[i]);  }  //returns vector of starting positions where pattern was found  vector<int> KMP() {  computeF();  int len = 0;  vector<int> res;  for (int i = 0; str[i]; ++i) {  len = getNewLen(len, str[i]);  if (!pat[len]) res.push\_back(i - len + 1);  }  return res;  } |

|  |
| --- |
| /\*  Number Theory Basics  \*/  #include <bits/stdc++.h>  using namespace std;  typedef long long ll;  //Doesn't sort  vector<int> divisors(int n) {  vector<int> v;  int i;  for (i = 1; i <= n / i; ++i) {  if (n % i == 0 && i\*i != n) {  v.push\_back(i);  v.push\_back(n / i);  }  if (i\*i == n) v.push\_back(i);  }  return v;  }  bool isPrime(int n) {  for (int i = 2; i <= n / i; i += 1 + (i & 1) )  if (n%i == 0) return 0;  return n > 1;  }  //Prime count under N can be estimated by N / lnN  int prime[N];  void sieve() {  memset(prime, 1, sizeof prime);  prime[0] = prime[1] = 0;  for (int i = 2; i <= N / i; i += 1 + (i % 1)) {  if (prime[i])  for (int j = i \* i; j <= N; j += i)  prime[j] = 0;  }  }  vector<pair<int, int>> factorize(int n) {  vector<pair<int, int>> v;  for (int i = 2; i <= n / i; ++i) {  int e = 0;  while (n%i == 0) {  n /= i;  e++;  }  if (e) v.push\_back({ i,e });  }  if (n > 1) v.push\_back({ n,1 });  return v;  }  ll GCD(ll a, ll b) {  while (b) {  ll t = a % b;  a = b;  b = t;  }  return a;  }  ll LCM(ll a, ll b) { return a / GCD(a, b) \* b; }  ll MODN(ll a, ll b) {  return (a%b + (b < 0 ? -b : b)) % b;  }  //Solves aX + bY = GCD(a,b)  //returns GCD for convenience.  #define update(t0,t) tmp = t, t = t0 - (a / b) \* t, t0 = tmp  ll eGCD(ll a, ll b, ll &x0, ll &y0) {  ll tmp, y = x0 = 1, x = y0 = 0;  while (b)  update(x0, x), update(y0, y), update(a, b);  return a;  }  //Solves (aX + bY = c) as long as c == m \* GCD(a,b)  //returns solvable  //general form: (X = x + k\*b/g), (Y = y - k\*a/g)  //that means either y = y \* c + ((x \* (c - 1)) / b) \* a;  // or x = x \* c + ((y \* (c - 1)) / a) \* b;  bool LDE(ll a, ll b, ll &x, ll &y, ll c, ll &g) {  g = eGCD(a, b, x, y);  if (c % g) return false;  a /= g, b /= g, c /= g; //we can reduce all of them by dividing by the GCD    //we should multiply by (c/g) but we already divided by g so we will multiply by only c  //We will calculate a new value for x:  x = MODN(x \* c, b);  //This line is useless for CRT since we don't use y  //But it uses the two general form formula to calculate the corresponding value of y  y = (c - a \* x) / b;  x = (c - b \* y) / a;  return true;  }  //Merges (X % m1 = r1) and (X % m2 = r2) into (X % m = r)  //returns solvable  bool combine(ll &m1, ll &r1, ll m2, ll r2) {  ll x, y, g;  if (!LDE(m1, m2, x, y, r2 - r1, g)) return false;  //assert(MODN(x \* m1 + r1, m1 \* (m2 / g)) == MODN(- y \* m2 + r2, m1 \* (m2 / g)));  r1 = x \* m1 + r1;  m1 = m1 \* (m2 / g);  r1 = MODN(r1, m1);  return true;  }  //Chinese Remainder theorm (X % m[i] = r[i])  bool CRT(const ll m[], const ll r[], int n, ll &m0, ll &r0) {  m0 = m[0], r0 = r[0];  for (int i = 1; i < n; i++)  if (!combine(m0, r0, m[i], r[i])) return false;  return true;  }  //Alternative modInverse(x,p) = x^(phi(p)-1) if p and x are coprime  //Or modInverse(x,p) = x^(p-2) p is prime;  ll modInverse(ll &x, ll mod) {  ll res,y;  if (!LDE(x, mod, res, y, 1)) return -1;  return res;  }  ll power(ll x, ll p, ll m) {  p = p % (m - 1); //m must be prime  ll res = 1;  while (p) {  if (p & 1) res = (res \* x) % m;  x = (x\*x) % m;  p >>= 1;  }  return res;  }  ll countFactorialFactors(ll n, ll p) {  ll count = 0;  while (n)  count += (n /= p);    }  ll combmem[100][100];  ll Combination(int n, int k) {  if (n == k) return 1;  if (k == 0) return 1;  ll & ret = combmem[n][k];  if (ret != 0) return ret;  else return Combination(n - 1, k - 1) + Combination(n - 1, k);  }  //phi(p) = p-1 <- p is prime  ll phi(ll x) {  ll res = x;  for (ll i = 2; i <= x / i; i += 1 + (i & 1)) {  if (x % i) continue;  while (x % i == 0)  x /= i;    res -= res / i;  }  if (x != 1) res -= res / x;  return res;  }  //powers of ten mod 1e9 + 7  const int N = 2e5;  const ll MOD = 1e9 + 7;  ll TENS[2 \* N];  ll\* tens = TENS + N;  const ll tenth = 700000005;  void init\_tens() {  tens[-1] = 700000005;  for (int i = 0; i < 1e5; i++) {  tens[i] = (tens[i - 1] \* 10) % MOD;  }  for (int i = -1; i > -1e5; i--) {  tens[i] = (tens[i + 1] \* tens[-1]) % MOD;  }  }  void gauss() {  for (int i = 0; i < n; i++) {  // Search for maximum in this column  double maxEl = abs(A[i][i]);  int maxRow = i;  for (int k = i + 1; k < n; k++) {  if (abs(A[k][i]) > maxEl) {  maxEl = abs(A[k][i]);  maxRow = k;  }  }  // Swap maximum row with current row (column by column)  for (int k = i; k < n + 1; k++) {  double tmp = A[maxRow][k];  A[maxRow][k] = A[i][k];  A[i][k] = tmp;  }  // Make all rows below this one 0 in current column  for (int k = i + 1; k < n; k++) {  double c = -A[k][i] / A[i][i];  for (int j = i; j < n + 1; j++) {  if (i == j) {  A[k][j] = 0;  }  else {  A[k][j] += c \* A[i][j];  }  }  }  }  // Solve equation Ax=b for an upper triangular matrix A  vector<double> x(n);  for (int i = n - 1; i >= 0; i--) {  x[i] = A[i][n] / A[i][i];  for (int k = i - 1; k >= 0; k--) {  A[k][n] -= A[k][i] \* x[i];  }  }  }  //counts numbers between s and e (inclusive) where n % x == m  ll count\_mod\_x(ll x, ll s, ll e, ll m) {  return (e - s + x - ((m - s) % x + x) % x) / x;  }//integer root  ll root(ll x) {  ll b = 0, e = 1e18 + 1;  while (b < e) {  ll m = (b + e) / 2;  if (m \* m > x) e = m;  else b = m + 1;  }  return b - 1;  } |

|  |
| --- |
| /\*  Some Geometry code  \*/  #define \_USE\_MATH\_DEFINES  #include <bits/stdc++.h>  #ifndef M\_PI  const long double M\_PI = acos((long double)-1);  #endif  using namespace std;  typedef long long ll;  typedef complex<double> point;  #define polar(r,t) ((r)\*exp(point(0,(t))))  #define polardeg(r,a) polar((r),(t) \* M\_PI/360)  const int N = 2e6 + 5, M = 1e2 + 5, OO = 0x7f7f7f7f7f7f7f7f;  typedef long long ll;  typedef pair<point, point> segment;  template<class T>  T GCD(T a, T b) {  while (b) {  T t = a % b;  a = b;  b = t;  }  return a;  }  struct line {  ll a, b, c;  line(ll x1, ll y1, ll x2, ll y2) {  a = y2 - y1;  b = x1 - x2;  c = -a \* x1 - b \* y1;  auto g = abs(\_\_gcd(a, \_\_gcd(b, c)));  if (a < 0 || a == 0 && b < 0) g \*= -1;  a /= g;  b /= g;  c /= g;  }  bool operator < (const line &q) const {  if (a != q.a) return a < q.a;  else if (b != q.b) return b < q.b;  else return c < q.c;  }  }; |

|  |
| --- |
| long long str\_to\_int(string num)  {  long long new\_num = 0, c = 1, test, siz = num.size(), i;  for (i = siz - 1; i >= 0; --i)  {  test = num[i] - '0';  test \*= c;  c \*= 10;  new\_num += test;  }  return new\_num;  }  //for max  long long binaryS()  {  long long low = 0, high = 2000000100, mid;  int forDouble = 120;  while (low < high /\*forDouble--\*/)  {  mid = (low + high + 1) / 2;  if (ok(mid)) low = mid;  else if (!ok(mid)) high = mid - 1;  }  return low;   1. } } |

|  |  |
| --- | --- |
| //knapsack | //coin Exchange |
| int knapsack(int i = n-1, int rem = k){  if(rem < 0) return -OO;  if(i == -1) return 0;  int & ret = mem[i][rem];  if(~ret) return ret;  ret = solve(i-1, rem);  ret = max(ret, solve(i-1, rem-W[i])+P[i]);  return ret;  } | long long coinE(int last = n-1, int rem = value){  if(rem < 0) return 0;  if(rem == 0) return 1;  if(last < 0) return 0;  long long & ret = mem[last][rem];  if(~ret) return ret;  return ret = solve(last-1, rem) + solve(last, rem-V[last]);  } |

|  |
| --- |
| //bellman (no optimization)  bool bellman(int src) {  memset(best, OO, sizeof best);  best[src] = 0;  int x = n;  while (x--) {  for (int e = 0; e < ne; e++) {  int u = from[e], v = to[e], c = cst[e];  if (best[u] + c < best[v]) {  if (!x) return 0;  best[v] = best[u] + c;  }  }  }  return 1;  } |

|  |
| --- |
| //maxflow shit  int vis[N], vid, q[N], qsz, best[N];  bool bfs() {  ++vid;  qsz = 0;  q[qsz++] = src;  memset(best, OO, n \* sizeof best[0]);  best[src] = 0;  vis[src] = vid;  for (int k = 0; k < qsz; ++k) {  int u = q[k];  for (int e = head[u], v; v = to[e], ~e; e = nxt[e]) {  if (vis[v] == vid || cap[e] == 0) continue;  vis[v] = vid;  best[v] = best[u] + 1;  if (v == snk) return 1;  q[qsz++] = v;  }  }  return 0;  }  int flowdfs(int u, int mn) {  if (u == snk)  return mn;  if (mn == 0 || vis[u] == vid)  return 0;  vis[u] = vid;  for (int &e = work[u]; ~e; e = nxt[e]) {  int v = to[e];  if (best[v] != best[u] + 1 )  continue;  int f = flowdfs(v, min(mn, cap[e]));  if (f) {  cap[e] -= f;  cap[e ^ 1] += f;  return f;  }  }  return 0;  }  int max\_flow() {  int f = 0;  while (bfs()) {  memcpy(work, head, n \* sizeof head[0]);  ++vid;  while (int chg = flowdfs(src, OO)) {  f += chg;  ++vid;  }  }  return f;  }  //or  ll best[N];  int parEdge[N];  bool bellman(int src) {  memset(best, OO, sizeof best);  best[src] = 0;  parEdge[src] = -1;  int x = N;  while (x--) {  for (int e = 0; e < ne; e++) {  if (!cap[e]) continue;  int u = to[e ^ 1], v = to[e];  ll c = cst[e];  if (best[u] + c < best[v]) {  if (!x) assert(0);  best[v] = best[u] + c;  parEdge[v] = e;  }  }  }  return best[snk] != OO;  }  ll augment() {  int e;  int u = snk;  int mn = OO;  while (~(e = parEdge[u])) {  mn = min(mn, cap[e]);  u = to[e ^ 1];  }  u = snk;  ll price = 0;  while (~(e = parEdge[u])) {  cap[e] -= mn;  cap[e ^ 1] += mn;  price += mn \* cst[e];  u = to[e ^ 1];  }  return price;  }  ll flow() {  memcpy(cap, cap, sizeof cap);  ll f = 0;  while (bellman(src)) {  ll chg = augment();  f += chg;  }  return f;  } |

|  |
| --- |
| //Fast Fourier  #include <complex>  #include <valarray>  using namespace std;  typedef complex<long double> comp;  long double PI = acosl(-1.0L);  #define polar(x) polar(1.0L,x);  const int MX\_LOGN = 17, MX\_N = 1 << MX\_LOGN;  comp wk[MX\_N / 2][MX\_LOGN + 1];  int LOG2[MX\_N + 1];  void precompute() {  for (int log2n = 0; log2n <= MX\_LOGN; log2n++) {  int N = 1 << log2n;  LOG2[N] = log2n;  for (int k = 0; k < N / 2; k++) {  wk[k][log2n] = polar(2.0L \* PI \* k / N);  }  }  }  valarray<comp> FFT(const valarray<comp>& a, int neg = -1) {  static int dummy = (precompute(), 0);  int N = a.size();  if (N == 1) return a;  valarray<comp> even = a[slice(0, N / 2, 2)];  valarray<comp> odd = a[slice(1, N / 2, 2)];  even = FFT(even, neg);  odd = FFT(odd, neg);  valarray<comp> ret(N);  for (int k = 0; k < N / 2; k++) {  comp w = wk[k][LOG2[N]];  w = { w.real(),w.imag()\*neg };  ret[k] = even[k] + w \* odd[k];  ret[N / 2 + k] = even[k] - w \* odd[k];  }  return ret;  }  valarray<comp> IFFT(const valarray<comp>& a) {  return FFT(a, 1) / comp(a.size());  } |

|  |
| --- |
| //Lovely 2-sat shit  //reminder: each variable is added twice in sequence.  int dfst, visidx[N], lowlink[N], cmpid[N], cmpn;  int stck[N], stcksz = 0;  void sccdfs(int u) {  visidx[u] = lowlink[u] = dfst++;  stck[stcksz++] = u;  for (int e = head[u], v; v = to[e], ~e; e = nxt[e]) {  if (visidx[v] == -1) {  sccdfs(v);  lowlink[u] = min(lowlink[u], lowlink[v]);  }  else if (cmpid[v] == -1) { //not cross edge  lowlink[u] = min(lowlink[u], visidx[v]);  }  }  if (lowlink[u] == visidx[u]) {  while (stcksz--) {  cmpid[stck[stcksz]] = cmpn;  if (stck[stcksz] == u) break;  }  cmpn++;  }  }  void scc() {  memset(visidx, -1, n \* sizeof visidx[0]);  memset(lowlink, -1, n \* sizeof lowlink[0]);  memset(cmpid, -1, n \* sizeof cmpid[0]);  cmpn = dfst = stcksz = 0;  for (int i = 0; i < n; i++) {  if (visidx[i] == -1)  sccdfs(i);  }  }  int nodes[N];  int value[N];  bool twosat() {  scc();  memset(value, -1, n \* sizeof value[0]);  for (int i = 0; i < n; i += 2) {  if (cmpid[i] == cmpid[i ^ 1]) return 0;  }  iota(nodes, nodes + n, 0);  sort(nodes, nodes + n, [](int a, int b) {return cmpid[a] < cmpid[b]; });  for (int i = 0; i < n; i++) {  if (~value[nodes[i]]) continue;  value[nodes[i]] = 1;  value[nodes[i] ^ 1] = 0;  }  return 1;  }  void add\_or(int a, int b) {  add\_edge(a ^ 1, b);  add\_edge(b ^ 1, a);  } |

|  |
| --- |
| //running median  priority\_queue<int> low; priority\_queue<int, vector<int>, greater<int>> high;  int getMedian() {  if (low.empty() && high.empty()) return 0;  if (low.size() == high.size()) return high.top(); //change this depending on median  else if (low.size() > high.size()) return low.top();  else return high.top();  }  void push(int v) {  int med = getMedian();  if (v < med)  low.push(v);  else  high.push(v);  if (low.size() > high.size() + 1) {  high.push(low.top());  low.pop();  }  else if (high.size() > low.size() + 1) {  low.push(high.top());  high.pop();  }  } |

|  |
| --- |
| #include <ext/pb\_ds/assoc\_container.hpp> // Common file  #include <ext/pb\_ds/tree\_policy.hpp> // Including tree\_order\_statistics\_node\_update  using namespace \_\_gnu\_pbds;  using namespace pb\_ds;  typedef tree<int, null\_type, less<int>, rb\_tree\_tag, tree\_order\_statistics\_node\_update> ordered\_set;  t.find\_by\_order(k) //returns kth item (0-based)  t.order\_of\_key(v) //returns number of elements strictly less than v |