```
Data Structures
                                                        void readn(double& n) {
    String
                                                         ld m; readn(m); n = m;
    Math
                                                        void readn(float& n) {
  ld m; readn(m); n = m;
    Graph
    2D Geometry
                                                        void readn(string& s) {
                                                         char c; s = ""
    3D Geometry
                                                         while((c=getchar_unlocked())!=' '&&c!='\n')
    Optimization
                                                         s += c:
     General
                                                        bool readline(string& s) {
                                                         char c; s = ""
run.sh
                                                         while(c=getchar_unlocked()) {
g++ -g -02 -std=gnu++17 -static prog.cpp
./a.exe
                                                          if (c == '\n') return true;
if (c == EOF) return false;
                                                          s += c:
                                                         return false:
# compile and test all *.in and *.ans
g++ -g -02 -std=gnu++17 -static prog.cpp
                                                        void printn(unsigned int n) {
for i in *.in; do f=${i%.in}
                                                         if (n / 10) printn(n / 10);
 ./a.exe < $i > "$f.out"
diff -b -q "$f.ans" "$f.out"
                                                         putchar_unlocked(n \frac{1}{2} 10 + \frac{10}{1}):
done
                                                        void printn(int n) {
                                                         if (n < 0) { putchar_unlocked('-'); n*=-1; }
Header
                                                         printn((unsigned int)n);
// use better compiler options
#pragma GCC optimize("Ofast", "unroll-loops")
#pragma GCC target("avx2,fma")
// include everything
                                                        2 Algorithms
                                                        Min/Max Subarray
#include <bits/stdc++.h>
#include <bits/extc++.h>
#include <sys/resource.h>
                                                        // max - compare = a < b. reset = a < 0
                                                        // min - compare = a > b, reset = a > 0
 // namespaces
                                                       // returns {sum, {start, end}}
pair<int, pair<int, int>
using namespace std;
using namespace __gnu_cxx; // rope
                                                             ContiguousSubarray(int* a, int size,
using namespace __gnu_pbds; // tree/trie
                                                            bool(*compare)(int, int),
// common defines
                                                           bool(*reset)(int), int defbest = 0) {
#define fastio
                                                         int best = defbest, cur = 0, start = 0, end =
\rightarrow \quad ios\_base::sync\_with\_stdio(0);cin.tie(0);\\ \textit{\#define nostacklim rlimit}
                                                        RZ: getrlimit(3.&RZ):RZ.rlim cur=-
                                                          cur += a[i]:

☐ 1; setrlimit(3, CRZ);
#define DEBUG(v) cout«"DEBUG: "«#v«" =
                                                          if ((*compare)(best, cur)) { best = cur;
                                                        \rightarrow "\langle v \rangle \langle n' \rangle; #define ll long long
#define ull unsigned ll
#define i128 __int128
#define u128 unsigned i128
                                                         return {best, {start, end}}:
                                                        Quickselect
#define ld long double
                                                        #define QSNE -999999
// global variables
                                                        int partition(int arr[], int 1, int r)
mt19937 rng((uint32 t)chrono::steady

    _clock::now().time_since_epoch().count());
                                                         int x = arr[r], i = 1;
                                                        for (int j = 1; j <= r - 1; j++)

if (arr[j] <= x)

...swap(arr[i++], arr[j]);
Fast IO
void readn(unsigned int& n) {
 char c; n = 0;
while ((c=getchar_unlocked())!=' '&&c!='\n')
                                                         swap(arr[i], arr[r]);
                                                         return i;
  n = n * 10 + c - 0';
                                                        // find k'th smallest element in unsorted
void readn(int% n) {
   char c; n = 0; int s = 1;
   if ((c=getchar_unlocked())=='-') s = -1;
                                                        → array, only if all distinct
                                                        int gselect(int arr[], int 1, int r, int k)
 else n = c - '0';
                                                        if (!(k > 0 && k <= r - 1 + 1)) return QSNE;
swap(arr[1 + rng() % (r-1+1)], arr[r]);
 while ((c=getchar_unlocked())!=' '&&c!='\n')
 n = n * 10 + c - 0;

n *= s;
                                                         int pos = partition(arr, 1, r);
                                                         if (pos-l==k-1) return arr[pos];
void readn(ld& n) {
                                                         if (pos-1>k-1) return qselect(arr,1,pos-1,k); };
 char c; n = 0;
ld m = 0, o = 1; bool d = false; int s = 1;
if ((c=getchar_unlocked())=='-') s = -1;
                                                         return gselect(arr, pos+1, r, k-pos+1-1);
                                                        // TODO: compare against std::nth_element()
 else if (c == '.') d = true;
else n = c - '0':
                                                        Saddleback Search
 while ((c=getchar_unlocked())!=' '&&c!='\n')
                                                        // search for v in 2d array arr[x][y], sorted
 on both axis
                                                       pair<int, int> saddleback_search(int** arr,
                                                        \stackrel{\square}{\hookrightarrow} int x, int v, int v) {
```

General

Algorithms

else n = n * 10 + c - '0':

n = s * (n + m * o):

```
int i = x-1, j = 0;
 while (i >= 0 && j < y) {
  if (arr[i][j] == v) return {i, j};
  (arr[i][i] > v)? i-: j++;
 return \{-1, -1\};
Ternary Search
// < max, > min, or any other unimodal func
#define TERNCOMP(a,b) (a)<(b)
int ternsearch(int a, int b, int (*f)(int)) {
 while (b-a > 4) { int m = (a+b)/2:
  if (TERNCOMP((*f)(m), (*f)(m+1))) a = m;
  else b = m+1:
 for (int i = a+1; i <= b; i++)
if (TERNCOMP((*f)(a), (*f)(i)))
 return a:
#define TERNPREC 0.000001
double ternsearch (double a, double b, double
 c  (*f)(double)) {
  while (b-a > TERNPREC * 4) {
    double m = (a+b)/2;
  if (TERNCOMP((*f)(m), (*f)(m + TERNPREC))) a
  else b = m + TERNPREC;
 for (double i = a + TERNPREC: i <= b: i +=

→ TERNPREC

     if (TERNCOMP((*f)(a), (*f)(i)))
   a = i:
 return á;
3 Data Structures
Fenwick Tree
// Fenwick tree, array of cumulative sums -
 \hookrightarrow O(log n) updates. O(log n) gets
struct Fenwick {
int n; ll* tree;
 void update(int i, int val) {
  ++i;
  while (i <= n) {
   tree[i] += val;
   i += i & (-i);
 Fenwick(int size) {
  n = size;
  tree = new ll[n+1];
  for (int i = 1; i <= n; i++)
tree[i] = 0;
 Fenwick(int* arr, int size) : Fenwick(size) {
   for (int i = 0; i < n; i++)
     update(i, arr[i]);</pre>
  ~Fenwick() { delete[] tree; }
 11 operator[](int i) {
  if (i < 0 || i > n) return 0;
  11 \hat{sum} = 0;
  .while (i>0) -
   "sum += tree[i]:
   i -= i & (-i);
  return sum;
 11 getRange(int a, int b) { return
    operator[](b) - operator[](a-1); }
Hashtable
// similar to unordered map, but faster
struct chash {
  const uint64_t C = (11)(2e18 * M_PI) + 71;
 11 operator()(11 x) const { return
    _builtin_bswap64(x*C); }
```

```
int main() {
  gp_hash_table<11,int,chash>
 \rightarrow hashtable({},{},{},{},{},{},{1\left(16)}):
for (int i = 0; i < 100; i++)

hashtable[i] = 200+i;

if (hashtable.find(10) != hashtable.end())
  cout « hashtable[10];
Ordered Set
typedef tree<int,null_type,less<int>,rb_tree
     _tag,tree_order_statistics_node_update>
    ordered_set;
int main()
 ordered_set o_set;
o_set.insert(5); o_set.insert(1);
 → o set.insert(3);
 // get second smallest element
 cout « *(o_set.find_by_order(1)) « '\n';
 // number of elements less than k=4
 cout « o_set.order_of_key(4) « '\n';
Rope
// O(log n) insert, delete, concatenate
int main() {
 .// generate rope
 rope<int> v:
 for (int i = 0: i < 100: i++)
 v.push back(i);
  // move range to front
 rope<int> copy = v.substr(10, 10);
 v.erase(10, 10);
 v.insert(copy.mutable_begin(), copy);
 // print elements of rope
 for (auto it : v) cout « it « " ";
Segment Tree
//max(a,b), min(a,b), a+b, a*b, gcd(a,b), a~b
struct SegmentTree {
 typedef int T;
 static constexpr T UNIT = INT MIN;
 Tf(Ta, Tb) {
if (a == UNIT) return b;
  if (b == UNIT) return a:
  return max(a,b);
 int n; vector<T> s;
 SegmentTree(int n, T def=UNIT) : s(2*n, def),
 \rightarrow n(n) {}
 SegmentTree(vector<T> arr)
 SegmentTree(arr.size()) {
  for (int i=0;i<arr.size();i++)

    update(i,arr[i]);

 void update(int pos, T val) {
  for (s[pos += n] = val; pos /= 2;)
   s[pos] = f(s[pos * 2], s[pos*2+1]);
 T query(int b, int e) { // query [b, e)
  T ra = UNIT, rb = UNIT;
  for (b+=n, e+=n; b<e; b/=2, e/=2) {
    if (b % 2) ra = f(ra, s[b++]);
    if (e % 2) rb = f(s[-e], rb);
  return f(ra, rb):
   get(int p) { return query(p, p+1); }
```

```
Trie
                                                                                            if (s[cur].out.size())
                                                                                             for (auto end : s[cur].out)
toret.push back({i - end.first + 1,
typedef trie<string, null type,

→ trie string access traits<>,

                                                                                              end.second}):
 ..pat_trie_tag,
return toret:
int main() {
 // generate trie
 trie_type trie;
 for (int i = 0; i < 20; i++)
                                                                                       Boyer Moore
 trie.insert(to_string(i)); // true if new,
                                                                                       struct defint { int i = -1; };
\hookrightarrow false if old
                                                                                       vector<int> boyermoore(string txt, string pat)
 // print things with prefix "1"
 auto range = trie.prefix_range("1");
                                                                                         vector<int> toret; unordered map<char,
 for (auto it = range.first; it !=

→ defint> badchar:

\hookrightarrow range.second; i\bar{t}++)
                                                                                         int m = pat.size(), n = txt.size();
  cout « *it « '
                                                                                        for (int i = 0; i < m; i++) badchar[pat[i]].i
                                                                                        \rightarrow = i;
int s = 0:
4 String
                                                                                        while (s \leq n - m) { int j = m - 1;
Aho Corasick
                                                                                           while (j \ge 0 \&\& pat[j] == txt[s + j]) j-;
                                                                                           if (j < 0) {
// range of alphabet for automata to consider
// MAXC = 26. OFFC = 'a' if only lowercase
                                                                                            toret.push_back(s);
const int MAXC = 256;
const int OFFC = 0:
                                                                                            s += (s + m < n) ? m - badchar[txt[s + m < n]) ] % m - badchar[txt[s + m < n]) % m - badchar[t
                                                                                             m]].i : 1;
struct aho_corasick {
 struct state
                                                                                            s += \max(1, i - badchar[txt[s + i]].i):
  set<pair<int, int> out;
                                                                                        return toret;
   int fail; vector<int> go;
   state() : fail(-1), go(MAXC, -1) {}
                                                                                       English Conversion
 vector<state> s:
                                                                                       const string ones[] = {"", "one", "two",
 int id = 0;
                                                                                             "three", "four", "five", "six", "seven", "eight", "nine";
 aho_corasick(string* arr, int size) : s(1) {
  for (int i = 0: i < size: i++) {
  ..int cur = 0;
                                                                                       const string teens[] ={"ten", "eleven",
  for (int c : arr[i]) {
   if (s[cur].go[c-OFFC] == -1) {
      s[cur].go[c-OFFC] = s.size();
                                                                                              "twelve", "thirteen", "fourteen", "fifteen", "sixteen", "seventeen",
                                                                                              "eighteen", "nineteen"};
       s.push_back(state());
                                                                                       const string tens[] = {"twenty", "thirty",
                                                                                             "forty", "fifty", "sixty", "seventy", "eighty", "ninety");
      cur = s[cur].go[c-OFFC];
    s[cur].out.insert({arr[i].size(), id++});
                                                                                       const string mags[] = {"thousand", "million",
                                                                                              "billion", "trillion", "quadrillion", "quintillion", "sextillion",
   for (int c = 0; c < MAXC; c++)
if (s[0].go[c] == -1)
                                                                                             "septillion"};
      s[0].go[c] = 0;
                                                                                       string convert(int num, int carry) {
   queue<int> sq;
                                                                                        if (num < 0) return "negative " +
  for (int c = 0; c < MAXC; c++) {
    if (s[0].go[c] != 0) {
        s[s[0].go[c]].fail = 0;
                                                                                              convert(-num, 0);
                                                                                        if (num < 10) return ones[num];
if (num < 20) return teens[num % 10];
      sq.push(s[0].go[c]);
                                                                                               (num < 100) return tens[(num / 10) - 2] +
                                                                                               (num%10==0?"":" ") + ones[num % 10];
                                                                                              (num < 1000) return ones [num / 100]
   while (sq.size()) {
                                                                                              (num/100==0?"":" ") + "hundred" + (num%100==0?"":" ") + convert(num % 100,
  int e = sq.front(); sq.pop();
for (int c = 0; c < MAXC; c++) {
   if (s[e].go[c] != -1) {</pre>
                                                                                             0);
    int failure = s[e].fail;
while (s[failure].go[c] == -1)
                                                                                        return convert(num / 1000, carry + 1) + " " +
                                                                                              mags[carry] + " " + convert(num % 1000.
        failure = s[failure].fail;
failure = s[failure].go[c];
                                                                                             0);
         s[s[e].go[c]].fail = failure;
                                                                                       string convert(int num) {
         for (auto length : s[failure].out)
                                                                                        return (num == 0) ? "zero" : convert(num, 0);
        s[s[e].go[c]].out.insert(length);
        sq.push(s[e].go[c]);
                                                                                       Knuth Morris Pratt
                                                                                       vector<int> kmp(string txt, string pat) {
                                                                                              vector<int> toret;
                                                                                         int m = txt.length(), n = pat.length();
                                                                                         int next[n + 1];
 // list of {start pos, pattern id}
                                                                                         for (int i = 0; i < n + 1; i++)
 vector<pair<int, int> search(string text)
                                                                                          next[i] = 0;
                                                                                        for (int i = 1; i < n; i++) {
  int j = next[i + 1];
   vector<pair<int, int> toret;
   int cur = 0;
                                                                                           while (j > 0 && pat[j] != pat[i])
   for (int i = 0; i < text.size(); i++) {
   while (s[cur].go[text[i]-OFFC] == -1)
                                                                                          if = next[j];
if (j > 0 || pat[j] == pat[i])
     cur = s[cur].fail;
```

next[i + 1] = i + 1;

cur = s[cur].go[text[i]-OFFC];

```
for (int i = 0, j = 0; i < m; i++) {
 if (txt[i] == pat[j]) {
  .if (++j == n)
    .toret.push_back(i - j + 1);
 } else if (j > 0) {
    j = next[j];
    i-;
 return toret;
Longest Common Prefix
string lcp(string* arr, int n) {
 if (n == 0) return ""
sort(arr, arr + n);
string r = ""; int v = 0;
 while (v < arr[0].length() && arr[0][v] ==

    arr[n-1][v])
    r += arr[0][v++];

 return r;
Longest Common Subsequence
string lcs(string a, string b) {
 int m = a.length(), n = b.length();
 int L[m+1][n+1];
for (int i = 0; i <= m; i++) {
    for (int j = 0; j <= n; j++) {
        if (i == 0 || j == 0) L[i][j] = 0;
        else if (a[i-1] == b[j-1]) L[i][j] =
  \begin{array}{ll} \to & L[i-1][j-1]+1; \\ & \text{ ... else L[i][j] = max(L[i-1][j], L[i][j-1]);} \end{array} 
 // return L[m][n]; // length of lcs
 string out = "";
int i = m - 1, j = n - 1;
while (i >= 0 && j >= 0) {
    if (a[i] == b[j]) {
  out = a[i-] + out:
...j-;
...}
  else if (L[i][j+1] > L[i+1][j]) i-;
  else j-;
 return out;
Longest Common Substring
// l is array of palindrome length at that

    int manacher(string s, int* 1) {
 int n = s.length() * 2;
 for (int i = 0, j = 0, k; i < n; i += k, j =
 \rightarrow max(j-k, 0)) {
  while (i \ge j \&\& i + j + 1 < n \&\& s[(i-j)/2]]
 \rightarrow == s[(i+i+1)/2]) i++:
  for (k = 1; i >= k && j >= k && l[i-k] !=
    j-k; k++)
  l[i+k] = min(l[i-k], j-k);
 return *max element(1, 1 + n):
Subsequence Count
 // "banana", "ban" » 3 (ban, ba..n, b..an)
ull subsequences(string body, string subs) {
 int m = subs.length(), n = body.length();
 if (m > n) return 0:
ull** arr = new ull*[m+1];
for (int i = 0; i <= m; i++) arr[i] = new
 \rightarrow ull[n+1];
 for (int i = 1; i <= m; i++) arr[i][0] = 0;
for (int i = 0; i <= n; i++) arr[0][i] = 1;
 for (int i = 1; i <= m; i++)
 for (int j = 1; j \le n; j++)
```

```
...arr[i][j] = arr[i][j-1] + ((body[j-1] ==
\rightarrow subs[i-1])? arr[i-1][j-1] : 0);
 return arr[m][n]:
5 Math
Catalan Numbers
ull* catalan = new ull[1000000];
void genCatalan(int n, int mod) {
 catalan[0] = catalan[1] = 1;
for (int i = 2; i <= n; i++) {
  catalan[i] = 0;
for (int j = i - 1; j >= 0; j-) {
   catalan[i] += (catalan[j] * catalan[i-j-1])
   % mod;
if (catalan[i] >= mod)
catalan[i] -= mod;
// TODO: consider binomial coefficient method
Combinatorics (nCr, nPr)
// can optimize by precomputing factorials, and
\hookrightarrow fact[n]/fact[n-r]
ull nPr(ull n, ull r) {
ull v = 1;
 for (ull i = n-r+1; i <= n; i++)
 .v *= i;
return v:
ull nPr(ull n, ull r, ull m) {
 ull v = 1;
 for (ull i = n-r+1; i <= n; i++)
v = (v * i) % m;
 return v:
ull nCr(ull n, ull r) {
long double v = 1;
 for (ull i = 1; i <= r; i++)
 v = v * (n-r+i) /i;
return (ull)(v + 0.001);
// requires modulo math
// can optimize by precomputing mfac and

→ minv-mfac

ull nCr(ull n, ull r, ull m) {
 return mfac(n, m) * minv(mfac(k, m), m) % m *
    minv(mfac(n-k, m), m) % m;
Chinese Remainder Theorem
bool ecrt(ll* r, ll* m, int n, ll& re, ll& mo)
it
ill x, y, d; mo = m[0]; re = r[0];
for (int i = 1; i < n; i++) {
    d = egcd(mo, m[i], x, y);
    if ((r[i] - re) % d != 0) return false;
    x = (r[i] - re) / d * x % (m[i] / d);
    re += x * mo;</pre>
  mo = mo / d * m[i];
  re %= mo;
 re = (re + mo) % mo:
 return true;
Count Digit Occurences
/*count(n,d) counts the number of occurences of
\rightarrow a digit d in the range [0,n]*/
ll digit_count(ll n, ll d) {
n /= 10;
 return result;
11 count(11 n, 11 d) {
 if (n < 10) return (d > 0 && n >= d);
if (n % 10) != 9) return digit_count(n, d) +
\hookrightarrow count(n-1, d);
return 10*count(n/10, d) + (n/10) + (d > 0);
```

```
Discrete Logarithm
                                                        Farey Fractions
unordered map<int, int> dlogc:
                                                         // generate 0 <= a/b <= 1 ordered. b <= n
int discretelog(int a, int b, int m) {
                                                         // fareu(4) = 0/1 1/4 1/3 1/2 2/3 3/4 1/1
dlogc.clear();
                                                         // length is sum of phi(i) for i = 1 to n
11 n = sqrt(m)+1, an = 1;
                                                         vector<pair<int, int> farey(int n) {
  int h = 0, k = 1, x = 1, y = 0, r;
for (int i = 0; i < n; i++)
an = (an * a) % m;
                                                          vector<pair<int, int> v;
 11 c = an;
 for (int i = 1; i <= n; i++) {
    if (!dlogc.count(c)) dlogc[c] = i;
                                                          v.push_back({h, k});
                                                          r = (n-y)/k;
 c = (c * an) \% m;
                                                           y += r*k; x += r*h;
                                                         swap(x,h); swap(y,k);
x = -x; y = -y;
while (k > 1);
 c = b:
 for (int i = 0; i <= n; i++) {
if (dlogc.count(c)) return (dlogc[c] * n - i
                                                          v.push_back(\{1, 1\});
                                                          return v;
\rightarrow + m - 1) % (m-1);
c = (c * a) % m;
                                                         Fast Fourier Transform
return -1;
                                                         const double PI = acos(-1):
                                                         void fft(vector<cd>& a, bool invert) {
Euler Phi / Totient
                                                          int n = a.size();
int phi(int n) {
                                                          for (int i = 1, j = 0; i < n; i++) {
 int^r = n;
for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) r -= r / i;
    while (n % i == 0) n /= i;
                                                           .int bit = n \gg 1
                                                          for (; j & bit; bit >= 1) j ^= bit;
j ^= bit;
                                                          if (i < i) swap(a[i], a[i]):
 if (n > 1) r = r / n;
                                                          for (int len = 2; len <= n; len <= 1) {
    double ang = 2 * PI / len * (invert ? -1 :
 return r;
#define n 100000
ll phi[n+1];
                                                           cd wlen(cos(ang), sin(ang));
void computeTotient() {
                                                           for (int i = 0; i < n; i += len) {
for (int i=1; i<=n; i++) phi[i] = i;
                                                            .cd w(1);
 for (int p=2; p<=n; p++) {
                                                            for (int j = 0; j < len / 2; j++) {
    cd u = a[i+j], v = a[i+j+len/2] * w;
 if (phi[p] == p) {
phi[p] = p-1;

for (int i = 2*p; i<=n; i += p) phi[i] =
                                                             a[i+i] = u + v:
                                                            a[i+j+len/2] = u - v;
w *= wlen;
   (phi[i]/p) * (p-1);
                                                          if (invert)
                                                          for (auto\& x : a)
Factorials
                                                           x /= n:
// digits in factorial
#define kamenetsky(n) (floor((n * log10(n /
                                                         vector<int> fftmult(vector<int> const& a.
\hookrightarrow M_E)) + (log10(2 * M_PI * n) / 2.0)) + 1)
                                                          → vector<int> const& b) {
// approximation of factorial
#define stirling(n) ((n == 1) ? 1 : sqrt(2 *
                                                          vector<cd> fa(a.begin(), a.end()),

    fb(b.begin(), b.end());

\hookrightarrow M PI * n) * pow(n / M E, n))
                                                          int n = 1 \ll (32 - \_builtin\_clz(a.size() +
// natural log of factorial
#define lfactorial(n) (lgamma(n+1))

→ b.size() - 1));
fa.resize(n); fb.resize(n);
                                                          fft(fa, false); fft(fb, false);
Prime Factorization
                                                          for (int i = 0; i < n; i++) fa[i] *= fb[i];
// do not call directly
                                                          fft(fa, true);
ll pollard_rho(ll n, ll s) {
                                                         return toret:
 x = mult(x, x, n);
x = (x + s) % n;
if (x == y) return n;
                                                         Greatest Common Denominator
                                                         | | 11 egcd(| 11 a, | 11 b, | 11 & x, | 11 & y) {
 ll d = __gcd(max(x - y, y - x), n);

if (1 < d && d < n) return d;

if (++head == tail) y = x, tail «= 1;
                                                          if (b == 0) { x = 1; y = 0; return a; }
                                                          11 gcd = egcd(b, a \% b, x, y);
                                                          x \stackrel{\smile}{-} a / b * y;
                                                          .swap(x, y);
                                                          return gcd:
// call for prime factors
void factorize(ll n, vector<ll> &divisor) {
if (n == 1) return;
if (isPrime(n)) divisor.push_back(n);
                                                         Josephus Problem
                                                         // 0-indexed, arbitrary k
 else {
.11 d = n;
                                                         int josephus(int n, int k) {
                                                         if (n == 1) return 0;
if (k == 1) return n-1;
 while (\vec{d} >= n) d = pollard_rho(n, rand() %
(n - 1) + 1);
factorize(n / d, divisor);
factorize(d, divisor);
                                                          if (k > n) return (joséphus(n-1,k)+k)%n;
                                                          int res = josephus(n-n/k,k)-n\%k;
                                                          return res + ((res<0)?n:res/(k-1));
```

```
| / / fast case if k=2, traditional josephus
int josephus(int n) {
 return 2*(n-(1*(32-\_builtin\_clz(n)-1)));
 Least Common Multiple
#define lcm(a,b) ((a*b)/qcd(a,b))
Modulo Operations
#define MOD 1000000007
#define madd(a,b,m) (a+b-((a+b-m>=0)?m:0))
#define mult(a,b,m) ((ull)a*b%m)
#define msub(a,b,m) (a-b+((a<b)?m:0))
ll mpow(ll b, ll e, ll m) {
 11 x = 1;
 while (e > 0) {
    if (e % 2) x = (x * b) % m;
    b = (b * b) % m;
  e /= 2;
  return x % m:
ull mfac(ull n, ull m) {
ull f = 1;
  for (int i = n; i > 1; i-)
 f = (f * i) % m;
return f:
// if m is not guaranteed to be prime
ll minv(ll b, ll m) {
    ll x = 0, y = 0;
    if (egcd(b, m, x, y) != 1) return -1;
    return (x % m + m) % m;
11 mdiv_compmod(int a, int b, int m) {
 if (_gcd(b, m) != 1) return -1;
 return mult(a, minv(b, m), m);
// if m is prime (like 10^9+7)
11 mdiv_primemod (int a, int b, int m) {
 return mult(a, mpow(b, m-2, m), m);
Miller-Rabin Primality Test
// Miller-Rabin primality test - O(10 log^3 n)
bool isPrime(ull n) {
  if (n < 2) return false;
  if (n == 2) return true;</pre>
 if (n % 2 == 0) return false;
ull s = n - 1;
while (s % 2 == 0) s /= 2;
  for (int i = 0; i < 10; i++) {
  ull temp = s;
  ull a = rand() % (n - 1) + 1;
ull mod = mpow(a, temp, n);
while (temp!=n-1&&mod!=1&&mod!=n-1) {
    mod = mult(mod, mod, n):
  if (mod!=n-1&&temp%2==0) return false;
  return true:
Sieve of Eratosthenes
bitset<100000001> sieve;
// generate sieve - O(n log n)
void genSieve(int n) {
 void gensieve(int n) {
    sieve[0] = sieve[1] = 1;
    for (ull i = 3; i * i < n; i += 2)
        ...if (!sieve[i])
        ...for (ull j = i * 3; j <= n; j += i * 2)
        ...sieve[j] = 1;</pre>
// query sieve after it's generated - O(1)
bool querySieve(int n) {
  return n == 2 || (n % 2 != 0 && !sieve[n]);
Simpson's / Approximate Integrals
```

// integrate f from a to b, k iterations

 $// error \le (b-a)/18.0 * M * ((b-a)/2k)^{2}$

```
|// where M = max(abs(f```(x))) for x in [a,b]
// "f" is a function "double func(double x)"
double Simpsons(double a, double b, int k,

    double (*f)(double)) {
 double dx = (b-a)/(2.0*k), t = 0;
 for (int i = 0; i < k; i++)

. t += ((i==0)?1:2)*(*f)(a+2*i*dx) + 4 *
 \leftrightarrow (*f)(a+(2*i+1)*dx);
return (t + (*f)(b)) * (b-a) / 6.0 / k;
 Common Equations Solvers
 // ax^2 + bx + c = 0, find x
 vector < double > solve Eq (double a, double b,
 double c) {
 vector<double> r;
 double z = b * b - 4 * a * c;
  if (z == 0)
  r.push_back(-b/(2*a));
 else if (z > 0) {
   r.push_back((sqrt(z)-b)/(2*a));
  r.push back((sqrt(z)+b)/(2*a));
 \frac{1}{2} / ax^3 + bx^2 + cx + d = 0, find x
vector < double > solveEq (double a. double b.

    double c, double d) {
    vector double res;
}
  long double a1 = b/a, a2 = c/a, a3 = d/a;
 long double q = (a1*a1 - 3*a2)/9.0, sq =
 \rightarrow -2*sqrt(q);
 long double r = (2*a1*a1*a1 - 9*a1*a2 +
 \rightarrow 27*a3)/54.0;
long double z = r*r-q*q*q, theta;
  if (z \le 0) {
  theta = acos(r/sqrt(q*q*q));
res.push_back(sq*cos(theta/3.0) - a1/3.0);
   res.push_back(sq*cos((theta+2.0*PI)/3.0) -
  \rightarrow a1/3.0):
   res.push_back(sq*cos((theta+4.0*PI)/3.0) -
    a1/3.0);
  res.push_back(pow(sqrt(z)+fabs(r), 1/3.0));
res[0] = (res[0] + q / res[0]) *
 \rightarrow ((r<0)?1:-1) - a1 / 3.0:
 return res;
\frac{1}{1} linear diophantine equation ax + by = c,
\rightarrow find x and y
// infinite solutions of form x+k*b/g, y-k*a/g bool solveEq(ll a, ll b, ll c, ll &x, ll &y,

→ 11 &g) {
 g = egcd(abs(a), abs(b), x, y);
if (c % g) return false;
 x *= c / g * ((a < 0) ? -1 : 1);
 y *= c / g * ((b < 0) ? -1 : 1);
return true:
^{\prime}/^{\prime} m = # equations, n = # variables, a[m][n+1]
 \Rightarrow = coefficient matrix
// a[i][0]x + a[i][1]y + ... + a[i][n]z =
const double eps = 1e-7;
bool zero(double a) { return (a < eps) && (a >
vector < double > solve Eq (double **a, int m, int
 if (j != cur) swap(a[j], a[cur]);
     for (int sat = 0; sat < m; sat++) {
  if (sat == cur) continue;</pre>
       double num = a[sat][i] / a[cur][i];
  for (int sot = 0; sot <= n; sot++)
```

```
.a[sat][sot] -= a[cur][sot] * num;
                                                      |// places total weight of the mst in &total
    cur++:
    break;
 for (int j = cur; j < m; j++)
   if (!zero(a[j][n])) return vector<double>();
 vector<double> ans(n,0);
for (int i = 0, sat = 0; i < n; i++)
if (sat < m && !zero(a[sat][i]))
ans[i] = a[sat][n] / a[sat++][i];
 return ans;
    Graph
struct edge {
 int u,v,w;
 edge (int u, int v, int w) : u(u), v(v), w(w) {}
 edge (): u(0), v(0), w(0) {}
bool operator < (const edge &e1, const edge
bool operator > (const edge &e1, const edge
\rightarrow &e2) { return e1.w > e2.w; }
struct subset { int p, rank; };
                                                       Union Find
Eulerian Path
#define edge_list vector<edge>
#define adj sets vector<set<int>
struct EulerPathGraph {
                                                       void uf_union(subset* s, int x, int y) {
adj_sets graph; // actually indexes incident
 edge_list edges; int n; vector<int> indeg;
 EulerPathGraph(int n): n(n) {
 indeg = *(new vector<int>(n,0));
 graph = *(new adj_sets(n, set<int>()));
void add_edge(int u, int v) {
  graph[u].insert(edges.size());
                                                      #define point complex<double>
                                                      double dot(point a, point b) { return
  indeg[v]++;

    real(conj(a)*b); }

  edges.push_back(edge(u,v,0));
                                                      double cross(point a, point b) { return
                                                       \rightarrow imag(conj(a)*b); }
 bool eulerian_path(vector<int> &circuit) {
                                                      struct line { point a, b; };
  if(edges.size()==0) return false;
 stack<int> st;
int a[] = {-1, -1};
                                                      struct circle { point c; double r; };
                                                      struct triangle { point a, b, c; };
  for(int v=0; v < n; v++)
                                                      struct rectangle { point tl, br; };
  if(indeg[v]!=graph[v].size()) {
                                                      struct convex_polygon {
   bool b = indeg[v] > graph[v].size();
if (abs(((int))indeg[v])-((int))graph[v]
     .size())) > 1) return
    false;

\widetilde{a[b]} = v;
 return false;
  int s = (a[0]!=-1 \&\& a[1]!=-1 ? a[0] :
\rightarrow (a[0]==-1 && a[1]==-1 ? edges[0].u : -1));
  if(s==-1) return false;
while(!st.empty() || !graph[s].empty()) {
  if (graph[s].empty()) {
    circuit.push back(s); s = st.top();
\stackrel{\hookrightarrow}{\Rightarrow} st.pop(); }
                                                      #define sq(a) ((a)*(a))
    int w = edges[*graph[s].begin()].v;
                                                      double circumference(circle a) { return 2 *
    graph[s].erase(graph[s].begin());
                                                       \rightarrow a.r * M_PI; }
    st.push(s); s = w;
  circuit.push back(s);
                                                      double intersection(circle a, circle b) {
  return circuit.size()-1==edges.size();
Minimum Spanning Tree
// returns vector of edges in the mst
// graph[i] = vector of edges incident to
                                                       \rightarrow sq(a.r)) / (2 * b.r * d));
\rightarrow vertex i
```

greater < edge » pq;
 vector < edge > MST;

pq.size()!=0) {

marked[u] =

total += e.w;

return s[i].p;

return MST:

MST.push back(e):

priority_queue<edge, vector<edge>,

bitset<20001> marked; // change size as

for (edge ep : graph[0]) pq.push(ep);

int u = e.u, v = e.v, w = e.w;
if(marked[u] && marked[v]) continue;
else if(marked[u]) swap(u, v);

for(edge ep : graph[u]) pq.push(ep);

 \rightarrow yp; else { s[yp].p = xp; s[xp].rank++; }

2D Geometry

vector<point points; convex_polygon(triangle a) {

imag(a.br)});

imag(a.tl)});

points.push_back(a.br);

double d = abs(a.c - b.c);if (d <= b.r - a.r) return area(a);
if (d <= a.r - b.r) return area(b);</pre> if $(d \ge a.r + b.r)$ return 0;

 \Rightarrow sq(b.r)) / (2 * a.r * d));

.}:

points.push_back(a.a);

points.push_back(a.b);

points.push_back(a.c);

convex_polygon(rectangle a) { points.push_back(a.tl);

points.push_back({real(a.tl),

points.push_back({real(a.br),

double alpha = acos((sq(a.r) + sq(d) - acos((sq(a.r) + sq(a) - acos((sq(a) - sq(a)

double beta = acos((sq(b.r) + sq(d) -

while(MST.size()!=graph.size()-1 &&

edge e = pq.top(); pq.pop();

```
return sq(a.r) * (alpha - 0.5 * sin(2 *
// if returned vector has size != n-1, there is
                                                             alpha)) + sq(b.r) * (beta - 0.5 * sin(2 *
                                                            beta));
vector<edge> mst(vector<vector<edge> graph, 11
                                                        double intersection (rectangle a, rectangle b)
                                                         double x1 = max(real(a.tl), real(b.tl)), y1 =
                                                         → max(imag(a.tl), imag(b.tl));
                                                         double x2 = min(real(a.br), real(b.br)), y2 =
                                                         min(imag(a.br), imag(b.br));
return (x2 <= x1 || y2 <= y1) ? 0 :</pre>
                                                             (x2-x1)*(y2-y1);
                                                             3D Geometry
                                                        struct point3d {
                                                         double x, y, z;
                                                         point3d operator+(point3d a) const { return
                                                         \rightarrow {x+a.x, y+a.y, z+a.z}; }
                                                         point3d operator*(double a) const { return
                                                         \rightarrow {x*a, y*a, z*a}; }
                                                         point3d operator-() const { return {-x, -y,
                                                             -z}; }
                                                         point3d operator-(point3d a) const { return
                                                         \rightarrow *this + -a; }
int uf find(subset* s, int i) {
  if (s[i].p!= i) s[i].p = uf_find(s, s[i].p);
                                                         point3d operator/(double a) const { return
                                                         *this * (1/a); }
double norm() { return x*x + y*y + z*z; }
                                                         double abs() { return sqrt(norm()); }
                                                         point3d normalize() { return *this /
int xp = uf_find(s, x), yp = uf_find(s, y);
                                                            this->abs(); }
if (s[xp].rank > s[yp].rank) s[yp].p = xp;
else if (s[xp].rank < s[yp].rank) s[xp].p =
                                                        double dot(point3d a, point3d b) { return
                                                         \rightarrow a.x*b.x + a.y*b.y + a.z*b.z; }
                                                        point3d cross(point3d a, point3d b) { return
                                                             {a.y*b.z - a.z*b.y, a.z*b.x - a.x*b.z,}
                                                            a.x*b.y - a.y*b.x; }
                                                        struct line3d { point3d a, b; };
                                                        struct plane { double a, b, c, d; } // a*x +
                                                         \rightarrow b*y + c*z + d = 0
                                                        struct sphere { point3d c; double r; };
                                                        \#define sq(a) ((a)*(a))
                                                        #define c\bar{b}(a) ((a)*(a)*(a)
                                                        double surface(circle a) { return 4 * sq(a.r)
                                                         → * M PI; }
                                                        double volume(circle a) { return 4.0/3.0 *
                                                        \rightarrow cb(a.r) * M PI; }
                                                             Optimization
                                                         // SameNumberOfOneBits, next permutation
                                                        int snoob(int a) {
  int b = a & -a, c = a + b;
  return c | ((a ^ c) » 2) / b;
                                                         // example usage
                                                        int main() {
  char l1|| =
                                                         the main() | { '1', '2', '3', '4' | char 12[] = { 'a', 'b', 'c', 'd' } ; int d1 = 5, d2 = 4; // prints 12345abcd, 1234a5bcd, ...
                                                         int min = (1 \le d1) - 1, max = min \le d2;
                                                         for (int i = min; i <= max; i = snoob(i)) {
  int p1 = 0, p2 = 0, v = i;
                                                          while (p1 < d1 || p2 < d2) {
cout « ((v & 1) ? l1[p1++] : l2[p2++]);
double area(circle a) { return sq(a.r) * M PI;
                                                           v /= 2;
                                                          cout « '\n';
                                                        Powers
                                                        bool isPowerOf2(ll a) {
                                                         return a > 0 \&\& !(a \& a-1);
                                                        bool isPowerOf3(11 a) {
```

```
return a>0&&!(12157665459056928801ull%a);
bool isPower(ll a, ll b) {
  double x = log(a) / log(b);
return abs(x-round(x)) < 0.00000000001;
```