# 1 General

## run.sh

```
g++ -g -O2 -std=gnu++17 -static prog.cpp
./a.exe
```

## test.sh

```
# compile and test all *.in and *.ans
g++ -g -O2 -std=gnu++17 -static prog.cpp
for i in *.in; do
 f=${i%.in}
 ./a.exe < $i > "$f.out"
 diff -b -q "$f.ans" "$f.out"
done
```

## Header

```
// use better compiler options
#pragma GCC optimize("Ofast","unroll-loops")
#pragma GCC target("avx2,fma")
// include everything
#include <bits/stdc++.h>
#include <bits/extc++.h>
#include <sys/resource.h>
// namespaces
using namespace std;
using namespace __gnu_cxx; // rope
using namespace __gnu_pbds; // tree/trie
// common defines
#define fastio ios_base::sync_with_stdio(0);cin.tie(0);
#define nostacklim rlimit RZ;getrlimit(3,&RZ);RZ.rlim_cur=-1;setrlimit(3,&RZ);
#define DEBUG(v) cerr<<__LINE__<<": "<<#v<<" = "<<v<<'\n';
#define TIMER cerr<<1.0*clock()/CLOCKS_PER_SEC<<"s\n";
#define ll long long
#define ull unsigned ll
#define i128 __int128
#define u128 unsigned i128
#define ld long double
// global variables
mt19937 rng((uint32_t)chrono::steady_clock::now().time_since_epoch().count());
```

## Fast IO

```
#ifdef _WIN32
#define getchar_unlocked() _getchar_nolock()
#define putchar_unlocked(x) _putchar_nolock(x)
#endif
void read(unsigned int& n) {
 char c; n = 0;
 while ((c=getchar_unlocked())!=' '&&c!='\n')
  n = n * 10 + c - '0';
}
void read(int& n) {
 char c; n = 0; int s = 1;
 if ((c=getchar_unlocked())=='-') s = -1;
 else n = c - '0';
 while ((c=getchar_unlocked())!=' '&&c!='\n')
  n = n * 10 + c - '0';
 n *= s;
}
void read(ld& n) {
 char c; n = 0;
 ld m = 0, o = 1; bool d = false; int s = 1;
 if ((c=getchar_unlocked())=='-') s = -1;
 else if (c == '.') d = true;
 else n = c - '0';
 while ((c=getchar_unlocked())!=' '&&c!='\n') {
  if (c == '.') d = true;
  else if (d) { m=m*10+c-'0'; o*=0.1; }
  else n = n * 10 + c - '0';
 }
 n = s * (n + m * o);
}
void read(double& n) {
 ld m; read(m); n = m;
}
void read(float& n) {
 ld m; read(m); n = m;
}
void read(string& s) {
 char c; s = "";
 while((c=getchar_unlocked())!=' '&&c!='\n')
  s += c;
}
bool readline(string& s) {
 char c; s = "";
 while(c=getchar_unlocked()) {
  if (c == '\n') return true;
  if (c == EOF) return false;
  s += c;
 }
 return false;
}
void print(unsigned int n) {
 if (n / 10) print(n / 10);
 putchar_unlocked(n % 10 + '0');
}
void print(int n) {
 if (n < 0) { putchar_unlocked('-'); n*=-1; }
 print((unsigned int)n);
}
```

## Common Structs

```
// n-dimension vectors
// Vec<2, int> v(n, m) = arr[n][m]
// Vec<2, int> v(n, m, -1) default init -1
template<int D, typename T>
struct Vec : public vector<Vec<D-1, T>> {
  template<typename... Args>
  Vec(int n=0, Args... args) : vector<Vec<D-1, T>>(n, Vec<D-1, T>(args...)) {}
};
template<typename T>
struct Vec<1, T> : public vector<T> {
  Vec(int n=0, T val=T()) : vector<T>(n, val) {}
};
```

# 2 Algorithms

## Min/Max Subarray

```
// max - compare = a < b, reset = a < 0
// min - compare = a > b, reset = a > 0
// returns {sum, {start, end}}
pair<int, pair<int, int>> ContiguousSubarray(int* a, int size, bool(*compare)(int, int), bool(*reset)(int), int defbest = 0) {
 int best = defbest, cur = 0, start = 0, end = 0, s = 0;
 for (int i = 0; i < size; i++) {
  cur += a[i];
  if ((*compare)(best, cur)) { best = cur; start = s; end = i; }
  if ((*reset)(cur)) { cur = 0; s = i + 1; }
 }
 return {best, {start, end}};
}
```

## Quickselect

```
#define QSNE -999999
int partition(int arr[], int l, int r)
{
 int x = arr[r], i = l;
 for (int j = l; j <= r - 1; j++)
  if (arr[j] <= x)
   swap(arr[i++], arr[j]);
 swap(arr[i], arr[r]);
 return i;
}
// find k'th smallest element in unsorted array, only if all distinct
int qselect(int arr[], int l, int r, int k)
{
 if (!(k > 0 && k <= r - l + 1)) return QSNE;
 swap(arr[l + rng() % (r-l+1)], arr[r]);
 int pos = partition(arr, l, r);
 if (pos-l==k-1) return arr[pos];
 if (pos-l>k-1) return qselect(arr,l,pos-1,k);
 return qselect(arr, pos+1, r, k-pos+l-1);
}
// TODO: compare against std::nth_element()
```

## Saddleback Search

```
// search for v in 2d array arr[x][y], sorted on both axis
pair<int, int> saddleback_search(int** arr, int x, int y, int v) {
 int i = x-1, j = 0;
 while (i >= 0 && j < y) {
  if (arr[i][j] == v) return {i, j};
  (arr[i][j] > 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)))
   a = i;
 return a;
}
#define TERNPREC 0.000001
double ternsearch(double a, double b, double (*f)(double)) {
 while (b-a > TERNPREC * 4) {
  double m = (a+b)/2;
  if (TERNCOMP((*f)(m), (*f)(m + TERNPREC))) a = m;
  else b = m + TERNPREC;
 }
 for (double i = a + TERNPREC; i <= b; i += TERNPREC)
    if (TERNCOMP((*f)(a), (*f)(i)))
   a = i;
 return a;
}
```

# 3 Structures

## Fenwick Tree

```
// Fenwick tree, array of cumulative sums - 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]);
 }
 ~Fenwick() { delete[] tree; }
 ll operator[](int i) {
  if (i < 0 || i > n) return 0;
  ll sum = 0;
  ++i;
  while (i>0) {
   sum += tree[i];
   i -= i & (-i);
  }
  return sum;
 }
 ll getRange(int a, int b) { return operator[](b) - operator[](a-1); }
};
```

## Hashtable

```
// similar to unordered_map, but faster
struct chash {
 const uint64_t C = (ll)(2e18 * M_PI) + 71;
 ll operator()(ll x) const { return __builtin_bswap64(x*C); }
};
int main() {
 gp_hash_table<ll,int,chash> hashtable({},{},{},{},{1<<16});
 for (int i = 0; i < 100; i++)
  hashtable[i] = 200+i;
 if (hashtable.find(10) != hashtable.end())
  cout << hashtable[10];
}
```

## Ordered Set

```
template <typename T>
using oset = tree<T,null_type,less<T>,rb_tree_tag,tree_order_statistics_node_update>;
template <typename T, typename D>
using omap = tree<T,D,less<T>,rb_tree_tag,tree_order_statistics_node_update>;
int main()
{
 oset<int> 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));
 // number of elements less than k=4
 cout << ' ' << o_set.order_of_key(4) << '\n';
 // equivalent with ordered map
 omap<int,int> o_map;
 o_map[5]=1;o_map[1]=2;o_map[3]=3;
 cout << (*(o_map.find_by_order(1))).first;
 cout << ' ' << o_map.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;
 T f(T a, T b) {
  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), 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);
 }
 T get(int p) { return query(p, p+1); }
};
```

## Trie

```
typedef trie<string, null_type,
    trie_string_access_traits<>,
 pat_trie_tag, trie_prefix_search_node_update>
    trie_type;
int main() {
 // generate trie
 trie_type trie;
 for (int i = 0; i < 20; i++)
  trie.insert(to_string(i)); // true if new,
    false if old
 // print things with prefix "1"
 auto range = trie.prefix_range("1");
 for (auto it = range.first; it !=
    range.second; it++)
  cout << *it << " ";
}
```

# 4 Strings

## Aho Corasick

```
// range of alphabet for automata to consider
// MAXC = 26, OFFC = 'a' if only lowercase
const int MAXC = 256;
const int OFFC = 0;
struct aho_corasick {
 struct state
 {
  set<pair<int, int>> out;
  int fail; vector<int> go;
  state() : fail(-1), go(MAXC, -1) {}
 };
 vector<state> s;
 int id = 0;
 aho_corasick(string* arr, int size) : s(1) {
  for (int i = 0; i < size; i++) {
   int cur = 0;
   for (int c : arr[i]) {
    if (s[cur].go[c-OFFC] == -1) {
     s[cur].go[c-OFFC] = s.size();
     s.push_back(state());
    }
    cur = s[cur].go[c-OFFC];
   }
   s[cur].out.insert({arr[i].size(), id++});
  }
  for (int c = 0; c < MAXC; c++)
   if (s[0].go[c] == -1)
    s[0].go[c] = 0;
  queue<int> sq;
  for (int c = 0; c < MAXC; c++) {
   if (s[0].go[c] != 0) {
    s[s[0].go[c]].fail = 0;
    sq.push(s[0].go[c]);
   }
  }
  while (sq.size()) {
   int e = sq.front(); sq.pop();
   for (int c = 0; c < MAXC; c++)
    if (s[e].go[c] != -1) {
     int failure = s[e].fail;
     while (s[failure].go[c] == -1)
      failure = s[failure].fail;
     failure = s[failure].go[c];
     s[s[e].go[c]].fail = failure;
     for (auto length : s[failure].out)
      s[s[e].go[c]].out.insert(length);
     sq.push(s[e].go[c]);
    }
  }
 }
 // list of {start pos, pattern id}
 vector<pair<int, int>> search(string text)
 {
  vector<pair<int, int>> toret;
  int cur = 0;
  for (int i = 0; i < text.size(); i++) {
   while (s[cur].go[text[i]-OFFC] == -1)
    cur = s[cur].fail;
   cur = s[cur].go[text[i]-OFFC];
   if (s[cur].out.size())
    for (auto end : s[cur].out)
     toret.push_back({i - end.first + 1,
    end.second});
  }
  return toret;
 }
};
```

## Boyer Moore

```
struct defint { int i = -1; };
vector<int> boyermoore(string txt, string pat)
    {
 vector<int> toret; unordered_map<char, defint>
    badchar;
 int m = pat.size(), n = txt.size();
 for (int i = 0; i < m; i++) badchar[pat[i]].i
    = i;
 int s = 0;
 while (s <= n - m) {
  int j = m - 1;
  while (j >= 0 && pat[j] == txt[s + j]) j--;
  if (j < 0) {
   toret.push_back(s);
   s += (s + m < n) ? m - badchar[txt[s +
    m]].i : 1;
  } else
   s += max(1, j - badchar[txt[s + j]].i);
 }
 return toret;
}
```

## English Conversion

```
const string ones[] = {"", "one", "two",
    "three", "four", "five", "six", "seven",
    "eight", "nine"};
const string teens[] ={"ten", "eleven",
    "twelve", "thirteen", "fourteen",
    "fifteen", "sixteen", "seventeen",
    "eighteen", "nineteen"};
const string tens[] = {"twenty", "thirty",
    "forty", "fifty", "sixty", "seventy",
    "eighty", "ninety"};
const string mags[] = {"thousand", "million",
    "billion", "trillion", "quadrillion",
    "quintillion", "sextillion",
    "septillion"};
string convert(int num, int carry) {
 if (num < 0) return "negative " +
    convert(-num, 0);
 if (num < 10) return ones[num];
 if (num < 20) return teens[num % 10];
 if (num < 100) return tens[(num / 10) - 2] +
    (num%10==0?"":" ") + ones[num % 10];
 if (num < 1000) return ones[num / 100] +
    (num/100==0?"":" ") + "hundred" +
    (num%100==0?"":" ") + convert(num % 100,
    0);
 return convert(num / 1000, carry + 1) + " " +
    mags[carry] + " " + convert(num % 1000,
    0);
}
string convert(int num) {
 return (num == 0) ? "zero" : convert(num, 0);
}
```

## Knuth Morris Pratt

```
vector<int> kmp(string txt, string pat) {
  vector<int> toret;
 int m = txt.length(), n = pat.length();
 int next[n + 1];
 for (int i = 0; i < n + 1; i++)
  next[i] = 0;
 for (int i = 1; i < n; i++) {
  int j = next[i + 1];
  while (j > 0 && pat[j] != pat[i])
   j = next[j];
  if (j > 0 || pat[j] == pat[i])
   next[i + 1] = j + 1;
 }
 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 (array)

```
// longest common prefix of strings in array
string lcp(string* arr, int n, bool sorted =
    false) {
 if (n == 0) return "";
 if (!sorted) 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] =
    L[i-1][j-1]+1;
   else L[i][j] = max(L[i-1][j], L[i][j-1]);
  }
 }
 // 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
    index
int manacher(string s, int* l) {
 int n = s.length() * 2;
 for (int i = 0, j = 0, k; i < n; i += k, j =
    max(j-k, 0)) {
  while (i >= j && i + j + 1 < n && s[(i-j)/2]
    == s[(i+j+1)/2]) j++;
  l[i] = j;
  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(l, l + n);
}
```

## Cyclic Rotation (Lyndon)

```
// simple strings = smaller than its nontrivial
    suffixes
// lyndon factorization = simple strings
    factorized
// "abaaba" -> "ab", "aab", "a"
vector<string> duval(string s) {
 int n = s.length();
 vector<string> lyndon;
 for (int i = 0; i < n;) {
  int j = i+1, k = i;
  for (; j < n && s[k] <= s[j]; j++)
   if (s[k] < s[j]) k = i;
   else k++;
  for (; i <= k; i += j - k)
   lyndon.push_back(s.substr(i,j-k));
 }
 return lyndon;
}
// lexicographically smallest rotation
int minRotation(string s) {
 int n = s.length(); s += s;
 auto d = duval(s); int i = 0, a = 0;
 while (a + d[i].length() < n) a +=
    d[i++].length();
 while (i && d[i] == d[i-1]) a -=
    d[i--].length();
 return a;
}
```

## 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
    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 <= n; j++)
   arr[i][j] = arr[i][j-1] + ((body[j-1] ==
    subs[i-1])? arr[i-1][j-1] : 0);
 return arr[m][n];
}
```

## Suffix Array + LCP

```
struct SuffixArray {
 vector<int> sa, lcp;
 SuffixArray(string& s, int lim=256) {
  int n = s.length() + 1, k = 0, a, b;
  vector<int> x(begin(s), end(s)+1), y(n),
    ws(max(n, lim)), rank(n);
  sa = lcp = y;
  iota(begin(sa), end(sa), 0);
  for (int j = 0, p = 0; p < n; j = max(1, j *
    2), lim = p) {
   p = j; iota(begin(y), end(y), n - j);
   for (int i = 0; i < (n); i++)
    if (sa[i] >= j)
     y[p++] = sa[i] - j;
   fill(begin(ws), end(ws), 0);
   for (int i = 0; i < (n); i++) ws[x[i]]++;
   for (int i = 1; i < (lim); i++) ws[i] +=
    ws[i - 1];
```

```
    for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
    swap(x, y); p = 1; x[sa[0]] = 0;
    for (int i = 1; i < (n); i++) {
      a = sa[i - 1]; b = sa[i];
      x[b] = (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
    }
  }
  for (int i = 1; i < (n); i++) rank[sa[i]] = i;
  for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
    for (k && k--, j = sa[rank[i] - 1];
      s[i + k] == s[j + k]; k++);
  }
};
```

### Suffix Tree (Ukkonen's)

```
struct SuffixTree {
  enum { N = 50000, ALPHA = 26 };
  int toi(char c) { return c - 'a'; }
  int t[N][ALPHA],l[N],r[N],p[N],s[N],v=0,q=0,m=2;
  string a;
  void ukkadd(int i, int c) { suff:
    if (r[v]<=q) {
      if (t[v][c]==-1) { t[v][c]=m; l[m]=i;
        p[m++]=v; v=s[v]; q=r[v]; goto suff; }
      v=t[v][c]; q=l[v];
    }
    if (q==-1 || c==toi(a[q])) q++; else {
      l[m+1]=i; p[m+1]=m; l[m]=l[v]; r[m]=q;
      p[m]=p[v]; t[m][c]=m+1; t[m][toi(a[q])]=v;
      l[v]=q; p[v]=m; t[p[m]][toi(a[l[m]])]=m;
      v=s[p[m]]; q=l[m];
      while (q<r[m]) { v=t[v][toi(a[q])]; q+=r[v]-l[v]; }
      if (q==r[m]) s[m]=v; else s[m]=m+2;
      q=r[v]-(q-r[m]); m+=2; goto suff;
    }
  }
  SuffixTree(string a) : a(a) {
    fill(r,r+N,(int)(a).size());
    memset(s, 0, sizeof s);
    memset(t, -1, sizeof t);
    fill(t[1],t[1]+ALPHA,0);
    s[0]=1;l[0]=l[1]=-1;r[0]=r[1]=p[0]=p[1]=0;
    for(int i=0;i<a.size();i++) ukkadd(i,toi(a[i]));
  }
  // Longest Common Substring between 2 strings
  // returns {length, offset from first string}
  pair<int, int> best;
  int lcs(int node, int i1, int i2, int olen) {
    if (l[node] <= i1 && i1 < r[node]) return 1;
    if (l[node] <= i2 && i2 < r[node]) return 2;
    int mask=0, len=node?olen+(r[node]-l[node]):0;
    for(int c=0; c<ALPHA; c++) if (t[node][c]!=-1)
      mask |= lcs(t[node][c], i1, i2, len);
    if (mask==3)
      best=max(best,{len,r[node]-len});
    return mask;
  }
  static pair<int, int> LCS(string s, string t) {
    SuffixTree st(s+(char)('z'+1)+t+(char)('z'+2));
    st.lcs(0, s.size(), s.size()+t.size()+1, 0);
    return st.best;
  }
};
```

### String Utilities

```
void lowercase(string& s) {
  transform(s.begin(), s.end(), s.begin(), ::tolower);
}
void uppercase(string& s) {
  transform(s.begin(), s.end(), s.begin(), ::toupper);
}
void trim(string &s) {
  s.erase(s.begin(),find_if_not(s.begin(),s.end(),[](int c){return isspace(c);}));
  s.erase(find_if_not(s.rbegin(),s.rend(),[](int c){return isspace(c);}).base(),s.end());
}
vector<string> split(string& s, char token) {
    vector<string> v; stringstream ss(s);
    for (string e;getline(ss,e,token);)
        v.push_back(e);
    return v;
}
```

## 5 Greedy

### Interval Cover

```
// L,R = interval [L,R], in = {{l,r}, index}
// does not handle case where L == R
vector<int> intervalCover(double L, double R, vector<pair<pair<double,double>,int>> in) {
    int i = 0; pair<double,int> pos = {L,-1};
    vector<int> a;
    sort(begin(in), end(in));
    while (pos.first < R) {
        double cur = pos.first;
        while (i < (int)in.size() && in[i].first.first <= cur)
            pos = max(pos,{in[i].first.second,in[i].second}), i++;
        if (pos.first == cur) return {};
        a.push_back(pos.second);
    }
    return a;
}
```

## 6 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 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;
}
```

### Multinomials

```
ll multinomial(vector<int>& v) {
  ll c = 1, m = v.empty() ? 1 : v[0];
  for(int i = 1; i < v.size(); i++)
    for (int j = 0; j < v[i]; j++)
      c = c * ++m / (j+1);
  return c;
}
```

### Chinese Remainder Theorem

```
bool ecrt(ll* r, ll* m, int n, ll& re, ll& mo) {
  ll 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;
    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 a digit d in the range [0,n]*/
ll digit_count(ll n, ll d) {
  ll result = 0;
  while (n != 0) {
    result += ((n%10) == d ? 1 : 0);
    n /= 10;
  }
  return result;
}
ll count(ll n, ll d) {
  if (n < 10) return (d > 0 && n >= d);
  if ((n % 10) != 9) return digit_count(n, d) + count(n-1, d);
  return 10*count(n/10, d) + (n/10) + (d > 0);
}
```

### Discrete Logarithm

```
unordered_map<int, int> dlogc;
int discretelog(int a, int b, int m) {
  dlogc.clear();
  ll n = sqrt(m)+1, an = 1;
  for (int i = 0; i < n; i++)
    an = (an * a) % m;
  ll c = an;
  for (int i = 1; i <= n; i++) {
    if (!dlogc.count(c)) dlogc[c] = i;
    c = (c * an) % m;
  }
  c = b;
  for (int i = 0; i <= n; i++) {
    if (dlogc.count(c)) return (dlogc[c] * n - i + m - 1) % (m-1);
    c = (c * a) % m;
  }
  return -1;
}
```

### Euler Phi / Totient

```
int phi(int n) {
  int r = n;
  for (int i = 2; i * i <= n; i++) {
    if (n % i == 0) r -= r / i;
    while (n % i == 0) n /= i;
  }
  if (n > 1) r -= r / n;
  return r;
}
#define n 100000
ll phi[n+1];
void computeTotient() {
  for (int i=1; i<=n; i++) phi[i] = i;
  for (int p=2; p<=n; p++) {
    if (phi[p] == p) {
      phi[p] = p-1;
      for (int i = 2*p; i<=n; i += p) phi[i] = (phi[i]/p) * (p-1);
    }
  }
}
```

### Factorials

```
// digits in factorial
#define kamenetsky(n) (floor((n * log10(n / M_E)) + (log10(2 * M_PI * n) / 2.0)) + 1)
// approximation of factorial
#define stirling(n) ((n == 1) ? 1 : sqrt(2 * M_PI * n) * pow(n / M_E, n))
// natural log of factorial
#define lfactorial(n) (lgamma(n+1))
```

### Prime Factorization

```
// do not call directly
ll pollard_rho(ll n, ll s) {
  ll x, y;
  x = y = rand() % (n - 1) + 1;
  int head = 1, tail = 2;
  while (true) {
    x = mult(x, x, n);
    x = (x + s) % n;
    if (x == y) return n;
    ll d = __gcd(max(x - y, y - x), n);
    if (1 < d && d < n) return d;
    if (++head == tail) y = x, tail <<= 1;
  }
}
// call for prime factors
void factorize(ll n, vector<ll> &divisor) {
  if (n == 1) return;
  if (isPrime(n)) divisor.push_back(n);
  else {
    ll d = n;
    while (d >= n) d = pollard_rho(n, rand() % (n - 1) + 1);
    factorize(n / d, divisor);
    factorize(d, divisor);
  }
}
```

### Farey Fractions

```
// generate 0 <= a/b <= 1 ordered, b <= n
// farey(4) = 0/1 1/4 1/3 1/2 2/3 3/4 1/1
// length is sum of phi(i) for i = 1 to n
vector<pair<int, int>> farey(int n) {
  int h = 0, k = 1, x = 1, y = 0, r;
  vector<pair<int, int>> v;
  do {
    v.push_back({h, k});
    r = (n-y)/k;
    y += r*k; x += r*h;
    swap(x,h); swap(y,k);
    x = -x; y = -y;
  } while (k > 1);
  v.push_back({1, 1});
  return v;
}
```

### Fast Fourier Transform

```
#define cd complex<double>
const double PI = acos(-1);
void fft(vector<cd>& a, bool invert) {
  int n = a.size();
  for (int i = 1, j = 0; i < n; i++) {
    int bit = n >> 1;
    for (; j & bit; bit >>= 1) j ^= bit;
    j ^= bit;
    if (i < j) swap(a[i], a[j]);
  }
  for (int len = 2; len <= n; len <<= 1) {
```

```
  double ang = 2 * PI / len * (invert ? -1 : 1);
  cd wlen(cos(ang), sin(ang));
  for (int i = 0; i < n; i += len) {
   cd w(1);
   for (int j = 0; j < len / 2; j++) {
    cd u = a[i+j], v = a[i+j+len/2] * w;
    a[i+j] = u + v;
    a[i+j+len/2] = u - v;
    w *= wlen;
   }
  }
 }
 if (invert)
  for (auto& x : a)
   x /= n;
}
vector<int> fftmult(vector<int> const& a, vector<int> const& b) {
 vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
 int n = 1 << (32 - __builtin_clz(a.size() + b.size() - 1));
 fa.resize(n); fb.resize(n);
 fft(fa, false); fft(fb, false);
 for (int i = 0; i < n; i++) fa[i] *= fb[i];
 fft(fa, true);
 vector<int> toret(n);
 for (int i = 0; i < n; i++) toret[i] = round(fa[i].real());
 return toret;
}
```

## Greatest Common Denominator

```
ll egcd(ll a, ll b, ll& x, ll& y) {
 if (b == 0) { x = 1; y = 0; return a; }
 ll gcd = egcd(b, a % b, x, y);
 x -= a / b * y;
 swap(x, y);
 return gcd;
}
```

## Josephus Problem

```
// 0-indexed, arbitrary k
int josephus(int n, int k) {
 if (n == 1) return 0;
 if (k == 1) return n-1;
 if (k > n) return (josephus(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)/__gcd(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) {
 ll 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;
}
ll 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)
ll mdiv_primemod (int a, int b, int m) {
 return mult(a, mpow(b, m-2, m), m);
}
```

## Modulo Tetration

```
ll tetraloop(ll a, ll b, ll m) {
 if(b == 0 || a == 1) return 1;
 ll w = tetraloop(a,b-1,phi(m)), r = 1;
 for (;w;w/=2) {
  if (w&1) {
   r *= a; if (r >= m) r -= (r/m-1)*m;
  }
  a *= a; if (a >= m) a -= (a/m-1)*m;
 }
 return r;
}
int tetration(int a, int b, int m) {
 if (a == 0 || m == 1) return ((b+1)&1)%m;
 return tetraloop(a,b,m) % m;
}
```

## Matrix

```
template<typename T>
struct Mat : public Vec<2, T> {
 int w, h;
 Mat(int x, int y) : Vec<2, T>(x, y), w(x), h(y) {}
 static Mat<T> identity(int n) { Mat<T> m(n,n);
   for (int i=0;i<n;i++) m[i][i] = 1; return m; }
 Mat<T>& operator+=(const Mat<T>& m) {
  for (int i = 0; i < w; i++)
   for (int j = 0; j < h; j++)
    (*this)[i][j] += m[i][j];
  return *this;
 }
 Mat<T>& operator-=(const Mat<T>& m) {
  for (int i = 0; i < w; i++)
   for (int j = 0; j < h; j++)
    (*this)[i][j] -= m[i][j];
  return *this;
 }
 Mat<T> operator*(const Mat<T>& m) {
  Mat<T> z(w,m.h);
  for (int i = 0; i < w; i++)
   for (int j = 0; j < h; j++)
    for (int k = 0; k < m.h; k++)
     z[i][k] += (*this)[i][j] * m[j][k];
  return z;
 }
 Mat<T> operator+(const Mat<T>& m) { Mat<T> a=*this; return a+=m; }
 Mat<T> operator-(const Mat<T>& m) { Mat<T> a=*this; return a-=m; }
 Mat<T>& operator*=(const Mat<T>& m) { return *this = (*this)*m; }
 Mat<T> power(int n) {
  Mat<T> a = Mat<T>::identity(w),m=*this;
  for (;n;n/=2,m*=m) if (n&1) a *= m;
  return a;
 }
};
```

## Matrix Exponentiation

```
// F(n) = c[0]*F(n-1) + c[1]*F(n-2) + ...
// b is the base cases of same length c
ll matrix_exponentiation(ll n, vector<ll> c, vector<ll> b) {
 if (nth < b.size()) return b[nth-1];
 Mat<ll> a(c.size(), c.size()); ll s = 0;
 for (int i = 0; i < c.size(); i++) a[i][0] = c[i];
 for (int i = 0; i < c.size() - 1; i++) a[i][i+1] = 1;
 a = a.power(nth - c.size());
 for (int i = 0; i < c.size(); i++)
  s += a[i][0] * b[i];
 return s;
}
```

## Nimber Arithmetic

```
#define nimAdd(a,b) ((a)^(b))
ull nimMul(ull a, ull b, int i=6) {
  static const ull M[]={INT_MIN>>32,
    M[0]^(M[0]<<16), M[1]^(M[1]<<8), M[2]^(M[2]<<4), M[3]^(M[3]<<2),
    M[4]^(M[4]<<1)};
  if (i-- == 0) return a&b;
  int k=1<<i;
  ull s=nimMul(a,b,i), m=M[5-i],
    t=nimMul(((a^(a>>k))&m)|(s&~m),
    ((b^(b>>k))&m)|(m&(~m>>1))<<k, i);
  return ((s^t)&m)<<k|((s^(t>>k))&m);
}
```

## Permutation

```
// c = array size, n = nth perm, return index
vector<int> gen_permutation(int c, int n) {
 vector<int> idx(c), per(c), fac(c); int i;
 for (i = 0; i < c; i++) idx[i] = i;
 for (i = 1; i <= c; i++) fac[i-1] = n%i, n/=i;
 for (i = c - 1; i >= 0; i--)
  per[c-i-1] = idx[fac[i]],
  idx.erase(idx.begin() + fac[i]);
 return per;
}
// get what nth permutation of vector
int get_permutation(vector<int>& v) {
 int use = 0, i = 1, r = 0;
 for (int e : v) {
  r = r * i++ + __builtin_popcount(use & -(1<<e));
  use |= 1 << e;
 }
 return r;
}
```

## Permutation (string/multiset)

```
string freq2str(vector<int>& v) {
 string s;
 for (int i = 0; i < v.size(); i++)
  for (int j = 0; j < v[i]; j++)
   s += (char)(i + 'A');
 return s;
}
// nth perm of multiset, n is 0-indexed
string gen_permutation(string s, ll n) {
 vector<int> freq(26, 0);
 for (auto e : s) freq[e - 'A']++;
 for (int i = 0; i < 26; i++) if (freq[i] > 0) {
  freq[i]--; ll v = multinomial(freq);
  if (n < v) return (char)(i+'A') + gen_permutation(freq2str(freq), n);
  freq[i]++; n -= v;
 }
 return "";
}
```

## 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;
 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);
   temp *= 2;
  }
  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) {
 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;
}
// 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 <= (b-a)/18.0 * M * ((b-a)/2k)^4
// 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 * (*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> solveEq(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));
 }
 return r;
}
// 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 = -2*sqrt(q);
 long double r = (2*a1*a1*a1 - 9*a1*a2 + 27*a3)/54.0;
 long double z = r*r-q*q*q, theta;
 if (z <= 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) - a1/3.0);
  res.push_back(sq*cos((theta+4.0*PI)/3.0) - a1/3.0);
 }
 else {
  res.push_back(pow(sqrt(z)+fabs(r), 1/3.0));
  res[0] = (res[0] + q / res[0]) * ((r<0)?1:-1) - a1 / 3.0;
 }
 return res;
}
// linear diophantine equation ax + by = c, 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, ll &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;
}
// m = # equations, n = # variables, a[m][n+1]
 = coefficient matrix
// a[i][0]x + a[i][1]y + ... + a[i][n]z =
 a[i][n+1]
// find a solution of some kind to linear
 equation
const double eps = 1e-7;
bool zero(double a) { return (a < eps) && (a >
 -eps); }
vector<double> solveEq(double **a, int m, int
 n) {
 int cur = 0;
 for (int i = 0; i < n; i++) {
  for (int j = cur; j < m; j++) {
   if (!zero(a[j][i])) {
    if (j != cur) swap(a[j], a[cur]);
    for (int sat = 0; sat < m; sat++) {
     if (sat == cur) continue;
     double num = a[sat][i] / a[cur][i];
     for (int sot = 0; sot <= n; sot++)
      a[sat][sot] -= a[cur][sot] * num;
    }
    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;
}
// solve A[n][n] * x[n] = b[n] linear equation
// rank < n is multiple solutions, -1 is no
 solutions
// `alls` is whether to find all solutions, or
 any
const double eps = 1e-12;
int solveEq(Vec<2, double>& A, Vec<1, double>&
 b, Vec<1, double>& x, bool alls=false) {
 int n = A.size(), m = x.size(), rank = 0, br,
 bc;
 vector<int> col(m); iota(begin(col), end(col),
 0);
 for(int i = 0; i < n; i++) {
  double v, bv = 0;
  for(int r = i; r < n; r++)
   for(int c = i; c < n; c++)
    if ((v = fabs(A[r][c])) > bv)
     br = r, bc = c, bv = v;
  if (bv <= eps) {
   for(int j = i; j < n; j++)
    if (fabs(b[j]) > eps)
     return -1;
   break;
  }
  swap(A[i], A[br]);
  swap(b[i], b[br]);
  swap(col[i], col[bc]);
  for(int j = 0; j < n; j++)
   swap(A[j][i], A[j][bc]);
  bv = 1.0 / A[i][i];
  for(int j = (alls)?0:i+1; j < n; j++) {
   if (j != i) {
    double fac = A[j][i] * bv;
    b[j] -= fac * b[i];
    for(int k = i+1; k < m; k++)
     A[j][k] -= fac*A[i][k];
   }
  }
  rank++;
 }
 if (alls) for (int i = 0; i < m; i++) x[i] =
 -DBL_MAX;
 for (int i = rank; i--;) {
  bool isGood = true;
  if (alls)
   for (int j = rank; isGood && j < m; j++)
    if (fabs(A[i][j]) > eps)
     isGood = false;
  b[i] /= A[i][i];
  if (isGood) x[col[i]] = b[i];
  if (!alls)
   for(int j = 0; j < i; j++)
    b[j] -= A[j][i] * b[i];
 }
 return rank;
}
```

## Graycode Conversions

```
ull graycode2ull(ull n) {
 ull i = 0;
 for (; n; n = n >> 1) i ^= n;
 return i;
}
ull ull2graycode(ull n) {
 return n ^ (n >> 1);
}
```

## Unix/Epoch Time

```
// 0-indexed month/time, 1-indexed day
// minimum 1970, 0, 1, 0, 0, 0
ull toEpoch(int year, int month, int day, int
 hour, int minute, int second) {
 struct tm t; time_t epoch;
 t.tm_year = year - 1900; t.tm_mon = month;
 t.tm_mday = day; t.tm_hour = hour;
 t.tm_min = minute; t.tm_sec = second;
 t.tm_isdst = 0; // 1 = daylights savings
 epoch = mktime(&t);
 return (ull)epoch;
}
vector<int> toDate(ull epoch) {
 time_t e=epoch; struct tm t=*localtime(&e);
 return {t.tm_year+1900,t.tm_mon,t.tm_mday,t
 .tm_hour,t.tm_min,t.tm_sec};
}
int getWeekday(ull epoch) {
 time_t e=epoch; struct tm t=*localtime(&e);
 return t.tm_wday; // 0-6, 0 = sunday
}
int getDayofYear(ull epoch) {
 time_t e=epoch; struct tm t=*localtime(&e);
 return t.tm_yday; // 0-365
}
const int months[] =
 {31,28,31,30,31,30,31,31,30,31,30,31};
bool validDate(int year, int month, int day) {
    bool leap = !(year%(year%25?4:16));
    if (month >= 12) return false;
    return day <= months[month] + (leap &&
 month == 1);
}
```

## Theorems and Formulae

**Montmort Numbers** count the number of derangements (permutations where no element appears in its original position) of a set of size $n$. $!0 = 1$, $!1 = 0$, $!n = (n+1)(!(n-1)+!(n-2))$, $!n = n!\sum_{i=0}^{n}\frac{(-1)^i}{i!}$, $!n = [\frac{n!}{e}]$

---

In a partially ordered set, a chain is a subset of elements that are all comparable to eachother. An antichain is a subset where no two are comparable.

**Dilworth's theorem** states the size of a maximal antichain equals the size of a minimal chain cover of a partially ordered set $S$. The width of $S$ is the maximum size of an antichain in $S$, which is equal to the minimum number of chains needed to cover $S$, or the minimum number of chains such that all elements are in at least one chain.

---

**Rosser's Theorem** states the $nth$ prime number is greater than $n * ln(n)$ for $n > 1$.

---

**Nicomachi's Theorem** states $1^3 + 2^3 + ... + n^3 = (1 + 2 + ... + n)^2$ and is equivalent to $(n\frac{n+1}{2})^2$.

---

**Lagrange's Four Square Theorem** states every natural number is the sum of the squares of four non-negative integers. This is a special case of the **Fermat Polygonal Number Theorem** where every positive integer is a sum of at most $n$ $s$-gonal numbers. The $nth$ $s$-gonal number $P(s,n) = (s-2)\frac{n(n-1)}{2} + n$

# 7 Graphs

```
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
 &e2) { return e1.w < e2.w; }
bool operator > (const edge &e1, const edge
 &e2) { return e1.w > e2.w; }
struct subset { int p, rank; };
```

## Eulerian Path

```
#define edge_list vector<edge>
#define adj_sets vector<set<int>>
struct EulerPathGraph {
 adj_sets graph; // actually indexes incident
 edges
 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());
  indeg[v]++;
  edges.push_back(edge(u,v,0));
 }
 bool eulerian_path(vector<int> &circuit) {
  if(edges.size()==0) return false;
  stack<int> st;
  int a[] = {-1, -1};
  for(int v=0;v<n;v++) {
   if(indeg[v]!=graph[v].size()) {
    bool b = indeg[v] > graph[v].size();
    if (abs(((int)indeg[v])-((int)graph[v]
 .size())) > 1) return
 false;
    if (a[b] != -1) return false;
    a[b] = v;
   }
  }
  int s = (a[0]!=-1 && a[1]!=-1 ? a[0] :
 (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();
 st.pop(); }
   else {
    int w = edges[*graph[s].begin()].v;
    graph[s].erase(graph[s].begin());
    st.push(s); s = w;
   }
  }
  circuit.push_back(s);
  return circuit.size()-1==edges.size();
 }
};
```

## Floyd Warshall

```
const ll inf = 1LL << 62;
#define FOR(i,n) for (int i = 0; i < n; i++)
void floydWarshall(Vec<2, ll>& m) {
 int n = m.size();
 FOR(i,n) m[i][i] = min(m[i][i], 0LL);
 FOR(k,n) FOR(i,n) FOR(j,n) if (m[i][k] != inf
 && m[k][j] != inf) {
  auto newDist = max(m[i][k] + m[k][j], -inf);
  m[i][j] = min(m[i][j], newDist);
 }
 FOR(k,n) if (m[k][k] < 0) FOR(i,n) FOR(j,n)
  if (m[i][k] != inf && m[k][j] != inf)
 m[i][j] = -inf;
}
```

## Minimum Spanning Tree

```
// returns vector of edges in the mst
// graph[i] = vector of edges incident to
 vertex i
// places total weight of the mst in &total
// if returned vector has size != n-1, there is
 no MST
vector<edge> mst(vector<vector<edge>> graph,
 ll &total) {
 total = 0;
 priority_queue<edge, vector<edge>,
 greater<edge>> pq;
 vector<edge> MST;
 bitset<20001> marked; // change size as needed
 marked[0] = 1;
 for (edge ep : graph[0]) pq.push(ep);
 while(MST.size()!=graph.size()-1 &&
 pq.size()!=0) {
  edge e = pq.top(); pq.pop();
  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);
  marked[u] = 1;
  MST.push_back(e);
  total += e.w;
 }
 return MST;
}
```

## Union Find

```
int uf_find(subset* s, int i) {
 if (s[i].p != i) s[i].p = uf_find(s, s[i].p);
 return s[i].p;
}
void uf_union(subset* s, int x, int y) {
 int xp = uf_find(s, x), yp = uf_find(s, y);
 if (s[xp].rank > s[yp].rank) s[yp].p = xp;
 else if (s[xp].rank < s[yp].rank) s[xp].p =
 yp;
 else { s[yp].p = xp; s[xp].rank++; }
}
```

## 2D Grid Shortcut

```
#define inbound(x,n) (0<=x&&x<n)
#define fordir(x,y,n,m) for(auto[dx,dy]:dir)if
 (inbound(x+dx,n)&&inbound(y+dy,m))
const pair<int,int> dir[] =
 {{1,0},{0,1},{-1,0},{0,-1}};
```

# 8 2D Geometry

```
#define point complex<double>
#define EPS 0.0000001
#define sq(a) ((a)*(a))
#define cb(a) ((a)*(a)*(a))
double dot(point a, point b) { return real(conj(a)*b); }
double cross(point a, point b) { return imag(conj(a)*b); }
struct line { point a, b; };
struct circle { point c; double r; };
struct segment { point a, point b; };
struct triangle { point a, b, c; };
struct rectangle { point tl, br; };
struct convex_polygon {
  vector<point> points;
  convex_polygon(vector<point> points) : points(points) {}
  convex_polygon(triangle a) {
    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), imag(a.br)});
    points.push_back(a.br);
    points.push_back({real(a.br), imag(a.tl)});
  }
};
struct polygon {
  vector<point> points;
  polygon(vector<point> points) : points(points) {}
  polygon(triangle a) {
    points.push_back(a.a); points.push_back(a.b); points.push_back(a.c);
  }
  polygon(rectangle a) {
    points.push_back(a.tl);
    points.push_back({real(a.tl), imag(a.br)});
    points.push_back(a.br);
    points.push_back({real(a.br), imag(a.tl)});
  }
  polygon(convex_polygon a) {
    for (point v : a.points)
      points.push_back(v);
  }
};
// triangle methods
double area_heron(double a, double b, double c) {
  if (a < b) swap(a, b);
  if (a < c) swap(a, c);
  if (b < c) swap(b, c);
  if (a > b + c) return -1;
  return sqrt((a+b+c)*(c-a+b)*(c+a-b)*(a+b-c)/16.0);
}
// segment methods
double lengthsq(segment a) { return sq(real(a.a) - real(a.b)) + sq(imag(a.a) - imag(a.b)); }
double length(segment a) { return sqrt(lengthsq(a)); }
// circle methods
double circumference(circle a) { return 2 * a.r * M_PI; }
double area(circle a) { return sq(a.r) * M_PI; }
// rectangle methods
double width(rectangle a) { return abs(real(a.br) - real(a.tl)); }
double height(rectangle a) { return abs(imag(a.br) - real(a.tl)); }
double diagonal(rectangle a) { return sqrt(sq(width(a)) + sq(height(a))); }
double area(rectangle a) { return width(a) * height(a); }
double perimeter(rectangle a) { return 2 * (width(a) + height(a)); }
// check if `a` fit's inside `b`
// swap equalities to exclude tight fits
bool doesFitInside(rectangle a, rectangle b) {
  int x = width(a), w = width(b), y = height(a), h = height(b);
  if (x > y) swap(x, y);
  if (w > h) swap(w, h);
  if (w < x) return false;
  if (y <= h) return true;
  double a=sq(y)-sq(x), b=x*h-y*w, c=x*w-y*h;
  return sq(a) <= sq(b) + sq(c);
}
// polygon methods
// negative area = CCW, positive = CW
double area(polygon a) {
  double area = 0.0; int n = a.points.size();
  for (int i = 0, j = 1; i < n; i++, j = (j + 1) % n)
    area += (real(a.points[j]-a.points[i]))*(imag(a.points[j]+a.points[i]));
  return area / 2.0;
}
// get both unsigned area and centroid
pair<double, point> area_centroid(polygon a) {
  int n = a.points.size();
  double area = 0;
  point c(0, 0);
  for (int i = n - 1, j = 0; j < n; i = j++) {
    double v = cross(a.points[i], a.points[j]) / 2;
    area += v;
    c += (a.points[i] + a.points[j]) * (v / 3);
  }
  c /= area;
  return {area, c};
}
```

## Intersection

```
// -1 coincide, 0 parallel, 1 intersection
int intersection(line a, line b, point& p) {
  if (abs(cross(a.b - a.a, b.b - b.a)) > EPS) {
    p = cross(b.a - a.a, b.b - a.b) / cross(a.b - a.a, b.b - b.a) * (b - a) + a;
    return 1;
  }
  if (abs(cross(a.b - a.a, a.b - b.a)) > EPS) return 0;
  return -1;
}
// area of intersection
double intersection(circle a, circle b) {
  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);
  if (d >= a.r + b.r) return 0;
  double alpha = acos((sq(a.r) + sq(d) - sq(b.r)) / (2 * a.r * d));
  double beta = acos((sq(b.r) + sq(d) - sq(a.r)) / (2 * b.r * d));
  return sq(a.r) * (alpha - 0.5 * sin(2 * alpha)) + sq(b.r) * (beta - 0.5 * sin(2 * beta));
}
// -1 outside, 0 inside, 1 tangent, 2 intersection
int intersection(circle a, circle b, vector<point>& inter) {
  double d2 = norm(b.c - a.c), rS = a.r + b.r, rD = a.r - b.r;
  if (d2 > sq(rS)) return -1;
  if (d2 < sq(rD)) return 0;
  double ca = 0.5 * (1 + rS * rD / d2);
  point z = point(ca, sqrt(sq(a.r) / d2 - sq(ca)));
  inter.push_back(a.c + (b.c - a.c) * z);
  if (abs(imag(z)) > EPS) inter.push_back(a.c + (b.c - a.c) * conj(z));
  return inter.size();
}
// points of intersection
vector<point> intersection(line a, circle c) {
  vector<point> inter;
  c.c -= a.a;
  a.b -= a.a;
  point m = a.b * real(c.c / a.b);
  double d2 = norm(m - c.c);
  if (d2 > sq(c.r)) return 0;
  double l = sqrt((sq(c.r) - d2) / norm(a.b));
  inter.push_back(a.a + m + l * a.b);
  if (abs(l) > EPS) inter.push_back(a.a + m - l * a.b);
  return inter;
}
// area of intersection
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 : (x2-x1)*(y2-y1);
}
```

## Convex Hull

```
bool cmp(point a, point b) {
  if (abs(real(a) - real(b)) > EPS) return real(a) < real(b);
  if (abs(imag(a) - imag(b)) > EPS) return imag(a) < imag(b);
  return false;
}
convex_polygon convexhull(polygon a) {
  sort(a.points.begin(), a.points.end(), cmp);
  vector<point> lower, upper;
  for (int i = 0; i < a.points.size(); i++) {
    while (lower.size() >= 2 &&
        cross(lower.back() - lower[lower.size() - 2], a.points[i] - lower.back()) < EPS)
      lower.pop_back();
    while (upper.size() >= 2 &&
        cross(upper.back() - upper[upper.size() - 2], a.points[i] - upper.back()) > -EPS)
      upper.pop_back();
    lower.push_back(a.points[i]);
    upper.push_back(a.points[i]);
  }
  lower.insert(lower.end(), upper.rbegin() + 1, upper.rend());
  return convex_polygon(lower);
}
```

# 9 3D Geometry

```
struct point3d {
  double x, y, z;
  point3d operator+(point3d a) const { return {x+a.x, y+a.y, z+a.z}; }
  point3d operator*(double a) const { return {x*a, y*a, z*a}; }
  point3d operator-() const { return {-x, -y, -z}; }
  point3d operator-(point3d a) const { return *this + -a; }
  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 / this->abs(); }
};
double dot(point3d a, point3d b) { return 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 + b*y + c*z + d = 0
struct sphere { point3d c; double r; };
#define sq(a) ((a)*(a))
#define cb(a) ((a)*(a)*(a))
double surface(circle a) { return 4 * sq(a.r) * M_PI; }
double volume(circle a) { return 4.0/3.0 * cb(a.r) * M_PI; }
```

# 10 Optimization

## Snoob

```
// 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[] = {'1', '2', '3', '4', '5'};
  char l2[] = {'a', 'b', 'c', 'd'};
  int d1 = 5, d2 = 4;
  // prints 12345abcd, 1234a5bcd, ...
  int min = (1<<d1)-1, max = min << 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++]);
      v /= 2;
    }
    cout << '\n';
  }
}
```

## Powers

```
bool isPowerOf2(ll a) {
  return a > 0 && !(a & a-1);
}
bool isPowerOf3(ll 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;
}
```

# 11 Additional

## Judge Speed

```
// kattis: 0.50s
// codeforces: 0.421s
// atcoder: 0.455s
#include <bits/stdc++.h>
using namespace std;
int v = 1e9/2, p = 1;
int main() {
  for (int i = 1; i <= v; i++) p *= i;
  cout << p;
}
```

## Judge Pre-Contest Checks

- __int128 and __float128 support?
- does extra or missing whitespace cause WA?
- documentation up to date?
- printer usage available and functional?

```cpp
// each case tests a different fail condition
// try them before contests to see error codes
struct g { int arr[1000000]; g(){}};
vector<g> a;
// 0=WA 1=TLE 2=MLE 3=OLE 4=SIGABRT 5=SIGFPE
↪  6=SIGSEGV 7=recursive MLE
int judge(int n) {
 if (n == 0) exit(0);
 if (n == 1) while(1);
 if (n == 2) while(1) a.push_back(g());
 if (n == 3) while(1) putchar_unlocked('a');
 if (n == 4) assert(0);
 if (n == 5) 0 / 0;
 if (n == 6) *(int*)(0) = 0;
 return n + judge(n + 1);
}
```

## GCC Builtin Docs

```cpp
// 128-bit integer
__int128 a;
unsigned __int128 b;
// 128-bit float
// minor improvements over long double
__float128 c;
// log2 floor
__lg(n);
// number of 1 bits
// can add ll like popcountll for long longs
__builtin_popcount(n);
// number of trailing zeroes
__builtin_ctz(n);
// number of leading zeroes
__builtin_clz(n);
// 1-indexed least significant 1 bit
__builtin_ffs(n);
// parity of number
__builtin_parity(n);
```

## Limits

| | | | |
|---|---|---|---|
| int | ±2147483647 | $\pm 2^{31}-1$ | $10^9$ |
| uint | 4294967295 | $2^{32}-1$ | $10^9$ |
| ll | ±9223372036854775807 | $\pm 2^{63}-1$ | $10^{18}$ |
| ull | 18446744073709551615 | $2^{64}-1$ | $10^{19}$ |
| i128 | ±170141183460469231... | $\pm 2^{127}-1$ | $10^{38}$ |
| u128 | 340282366920938463... | $2^{128}-1$ | $10^{38}$ |

Complexity classes input size (per second):

| | |
|---|---|
| $O(n^n)$ or $O(n!)$ | $n \leq 10$ |
| $O(2^n)$ | $n \leq 30$ |
| $O(n^3)$ | $n \leq 1000$ |
| $O(n^2)$ | $n \leq 30000$ |
| $O(n\sqrt{n})$ | $n \leq 10^6$ |
| $O(n \log n)$ | $n \leq 10^7$ |
| $O(n)$ | $n < 10^9$ |