南京大学 ACM-ICPC 集训队代码模版库

Contents

1 General

1.1 Code library checksum

```
ab14 #!/usr/bin/python3
c502 import re, sys, hashlib
427e
f7db for line in sys.stdin.read().strip().split("\n") :
ddf5     print(hashlib.md5(re.sub(r'\s|//.*', '', line).encode('utf8')).hexdigest()
           [-4:], line)
```

1.2 Makefile

```
dab2 .PHONY : run
427e
5f25 run : $(t)
427e
207e $(t) : $(t).cpp
155d   g++ --std=c++14 -O2 -D__LOCAL_DEBUG__ -fsanitize=undefined -fsanitize=address
         -ggdb -pipe -o $@ $<
```

1.3 .vimrc

```
914c set nocompatible
733d syntax on
6bbc colorscheme slate
7db5 set number
b0e3 set cursorline
061b set shiftwidth=2
8011 set softtabstop=2
a66d set tabstop=2
d23a set expandtab
5245 set magic
740c set smartindent
bee8 set backspace=indent,eol,start
815d set cmdheight=1
0a40 set laststatus=2
e458 set statusline=\ %<%F[%1*%M%*%n%R%H]%=\ %y\ %0(%{&fileformat}\ %{&encoding}\ %c
       :%l/%L%)\
1c67 set whichwrap=b,s,<,>,[,]
```

1.4 Template

```
302f #include <bits/stdc++.h>
421c using namespace std;
427e
426f #ifdef __LOCAL_DEBUG__
3341 # define _debug(fmt, ...) fprintf(stderr, "[%s]␣" fmt "\n", \
611f     __func__, ##__VA_ARGS__)
a8cb #else
e6b5 # define _debug(...) ((void) 0)
1937 #endif
0d6c #define rep(i, n) for (int i=0; i<(n); i++)
cfe3 #define Rep(i, n) for (int i=1; i<=(n); i++)
8843 #define range(x) (x).begin(), (x).end()
5cad typedef long long LL;
b773 typedef unsigned long long ULL;
427e
5120 template <unsigned p>
87b8 struct Zp{
7797     unsigned x;
ff67     Zp(unsigned x):x(x){}
22e3     operator unsigned(){return x;}
fecc     Zp operator ^ (ULL e) {
4fce         Zp b=x, r=1;
3e90         while (e) {
5421             if (e&1) r=r*b;
2059             b=b*b;
16fc             e>>=1;
95cf         }
547e         return r;
95cf     }
a2f5     Zp operator + (Zp rhs) {return (x+rhs)%p;}
664b     Zp operator - (Zp rhs) {return (x+p-rhs)%p;}
3ec4     Zp operator * (Zp rhs) {return x*rhs%p;}
7cfd     Zp operator / (Zp rhs) {return Zp(x)*(rhs^(p-2));}
329b };
427e
370f typedef Zp<1000000007> zp;
427e
0795 zp operator"" _ (ULL n){return n;}
```

2 Miscellaneous Algorithms

2.1 2-SAT

```
0f42 const int MAXN = 100005;
03a9 struct twoSAT{
5c83     int n;
8f72     vector<int> G[MAXN*2];
d060     bool mark[MAXN*2];
b42d     int S[MAXN*2], c;
427e
d34f     void init(int n){
b985         this->n = n;
f9ec         for (int i=0; i<n*2; i++) G[i].clear();
0609         memset(mark, 0, sizeof(mark));
95cf     }
427e
3bd5     bool dfs(int x){
bd70         if (mark[x^1]) return false;
c96a         if (mark[x]) return true;
fd23         mark[x] = true;
4bea         S[c++] = x;
1ce6         for (int i=0; i<G[x].size(); i++)
d942             if (!dfs(G[x][i])) return false;
3361         return true;
95cf     }
427e
5894     void add_clause(int x, bool xval, int y, bool yval){
6afe         x = x * 2 + xval;
e680         y = y * 2 + yval;
81cc         G[x^1].push_back(y);
6835         G[y^1].push_back(x);
95cf     }
427e
d0cb     bool solve() {
7c39         for (int i=0; i<n*2; i+=2){
e63f             if (!mark[i] && !mark[i+1]){
88fb                 c = 0;
f4b9                 if (!dfs(i)){
3f03                     while (c > 0) mark[S[--c]] = false;
86c5                     if (!dfs(i+1)) return false;
95cf                 }
95cf             }
95cf         }
95cf         return true;
95cf     }
427e
5f0a     inline bool value(unsigned i){return mark[2*i+1];}
329b };
```

2.2 Knuth's optimization

```
5c83 int n;
d77c int dp[256][256], dc[256][256];
427e
b7ec template <typename T>
0bc7 void compute(T cost) {
0423   for (int i = 0; i <= n; i++) {
8f5e     dp[i][i] = 0;
9488     dc[i][i] = i;
95cf   }
be8e   rep (i, n) {
95b5     dp[i][i+1] = 0;
aa0f     dc[i][i+1] = i;
95cf   }
ec08   for (int len = 2; len <= n; len++) {
88b8     for (int i = 0; i + len <= n; i++) {
d3da       int j = i + len;
9824       int lbnd = dc[i][j-1], rbnd = dc[i+1][j];
a24a       dp[i][j] = INT_MAX / 2;
f933       int c = cost(i, j);
90d2       for (int k = lbnd; k <= rbnd; k++) {
9bd0         int res = dp[i][k] + dp[k][j] + c;
26b5         if (res < dp[i][j]) {
e6af           dp[i][j] = res;
9c88           dc[i][j] = k;
95cf         }
95cf       }
95cf     }
95cf   }
329b };
```

2.3 Mo's algorithm

All intervals are closed on both sides. When running functions `enter()` and `leave()`, the global l and r has not changed yet.

Usage:

`add_query(id, l, r)`	Add id-th query $[l, r]$.
`run()`	Run Mo's algorithm.
`init()`	**TODO**. Initialize the range $[l, r]$.
`yield(id)`	**TODO**. Yield answer for id-th query.
`enter(o)`	**TODO**. Add o-th element.
`leave(o)`	**TODO**. Remove o-th element.

```
5194 constexpr int BLOCK_SZ = 300;
427e
3ec4 struct query { int l, r, id; };
d26a vector<query> queries;
427e
1e30 void add_query(int id, int l, int r) {
54c9   queries.push_back(query{l, r, id});
95cf }
427e
9f6b int l, r;
427e
427e // ----- functions to implement -----
62b4 inline void init();
50e1 inline void yield(int id);
b20d inline void enter(int o);
13af inline void leave(int o);
427e
37f0 void run() {
ab0b   if (queries.empty()) return;
8508   sort(range(queries), [](query lhs, query rhs) {
c7f8     int lb = lhs.l / BLOCK_SZ, rb = rhs.l / BLOCK_SZ;
03e7     if (lb != rb) return lb < rb;
0780     return lhs.r < rhs.r;
b251   });
6196   l = queries[0].l;
9644   r = queries[0].r;
07e2   init();
5bc9   for (query q : queries) {
7bc7     while (l > q.l) enter(l - 1), l--;
d646     while (r < q.r) enter(r + 1), r++;
13f0     while (l < q.l) leave(l), l++;
e1c6     while (r > q.r) leave(r), r--;
82f5     yield(q.id);
95cf   }
95cf }
```

3 String

3.1 Knuth-Morris-Pratt algorithm

```
2836 const int SIZE = 10005;
427e
d02b struct kmp_matcher {
2d81   char p[SIZE];
9847   int fail[SIZE];
57b7   int len;
427e
60cf   void construct(const char* needle) {
aaa1     len = strlen(p);
3a87     strcpy(p, needle);
3dd4     fail[0] = fail[1] = 0;
d8a8     for (int i = 1; i < len; i++) {
147f       int j = fail[i];
3c79       while (j && p[i] != p[j]) j = fail[j];
4643       fail[i + 1] = p[i] == p[j] ? j + 1 : 0;
95cf     }
95cf   }
427e
c464   inline void found(int pos) {
427e     // ! add codes for having found at pos
95cf   }
427e
2daf   void match(const char* haystack) {  // must be called after construct
700f     const char* t = haystack;
8482     int n = strlen(t);
8fd0     int j = 0;
be8e     rep(i, n) {
4e19       while (j && p[j] != t[i]) j = fail[j];
b5d5       if (p[j] == t[i]) j++;
f024       if (j == len) found(i - len + 1);
95cf     }
95cf   }
329b };
```

3.2 Manacher algorithm

```
81d4 struct Manacher {
cd09   int Len;
9255   vector<int> lc;
b301   string s;
427e
ec07   void work() {
c033     lc[1] = 1;
6bef     int k = 1;
427e
491f     for (int i = 2; i <= Len; i++) {
7957       int p = k + lc[k] - 1;
5e04       if (i <= p) {
24a1         lc[i] = min(lc[2 * k - i], p - i + 1);
8e2e       } else {
e0e5         lc[i] = 1;
95cf       }
74ff       while (s[i + lc[i]] == s[i - lc[i]]) lc[i]++;
2b9a       if (i + lc[i] > k + lc[k]) k = i;
95cf     }
95cf   }
427e
bfd5   void init(const char *tt) {
aaaf     int len = strlen(tt);
f701     s.resize(len * 2 + 10);
7045     lc.resize(len * 2 + 10);
8e13     s[0] = '*';
ae54     s[1] = '#';
1321     for (int i = 0; i < len; i++) {
e995       s[i * 2 + 2] = tt[i];
69fd       s[i * 2 + 1] = '#';
95cf     }
43fd     s[len * 2 + 1] = '#';
75d1     s[len * 2 + 2] = '\0';
61f7     Len = len * 2 + 2;
3e7a     work();
95cf   }
427e
b194   pair<int, int> maxpal(int l, int r) {
901a     int center = l + r + 1;
ffb2     int rad = lc[center] / 2;
ab54     int rmid = (l + r + 1) / 2;
17e4     int rl = rmid - rad, rr = rmid + rad - 1;
3908     if ((r ^ l) & 1) {
69f3     } else rr++;
69dc     return {max(l, rl), min(r, rr)};
95cf   }
329b };
```

3.3 Aho-corasick automaton

```
a1ad struct AC : Trie {
9143   int fail[MAXN];
daca   int last[MAXN];
427e
8690   void construct() {
93d2     queue<int> q;
a7a6     fail[0] = 0;
ce3c     rep(c, CHARN) {
b1c6       if (int u = tr[0][c]) {
a506         fail[u] = 0;
3e14         q.push(u);
f689         last[u] = 0;
95cf       }
95cf     }
cc78     while (!q.empty()) {
31f0       int r = q.front();
15dd       q.pop();
ce3c       rep(c, CHARN) {
ab59         int u = tr[r][c];
0ef5         if (!u) {
9d58           tr[r][c] = tr[fail[r]][c];
b333           continue;
95cf         }
3e14         q.push(u);
b3ff         int v = fail[r];
d2ea         while (v && !tr[v][c]) v = fail[v];
c275         fail[u] = tr[v][c];
654c         last[u] = tag[fail[u]] ? fail[u] : last[fail[u]];
95cf       }
95cf     }
95cf   }
427e
7752   void found(int pos, int j) {
```

```
043e      if (j) {
427e        // ! add codes for having found word with tag[j]
4a96        found(pos, last[j]);
95cf      }
95cf    }
427e
9785    void find(const char* text) {  // must be called after construct()
80a4      int p = 0, c, len = strlen(text);
9c94      rep(i, len) {
b3db        c = id(text[i]);
f119        p = tr[p][c];
f08e        if (tag[p])
389b          found(i, p);
1e67        else if (last[p])
299e          found(i, last[p]);
95cf      }
95cf    }
329b  };
```

3.4 Trie

```
e6f1  const int MAXN = 12000;
dd87  const int CHARN = 26;
427e
8ff5  inline int id(char c) { return c - 'a'; }
427e
a281  struct Trie {
5c83    int n;
f4f5    int tr[MAXN][CHARN];  // Trie tree, 0 denotes fail
35a5    int tag[MAXN];
427e
4fee    Trie() {
3ccc      memset(tr[0], 0, sizeof(tr[0]));
4d52      tag[0] = 0;
46bf      n = 1;
95cf    }
427e
427e    // tag should not be 0
30b0    void add(const char* s, int t) {
d50a      int p = 0, c, len = strlen(s);
9c94      rep(i, len) {
3140        c = id(s[i]);
d6c8        if (!tr[p][c]) {
26dd          memset(tr[n], 0, sizeof(tr[n]));
2e5c          tag[n] = 0;
73bb          tr[p][c] = n++;
95cf        }
f119        p = tr[p][c];
95cf      }
35ef      tag[p] = t;
95cf    }
427e
427e    // returns 0 if not found
427e    // AC automaton does not need this function
216c    int search(const char* s) {
d50a      int p = 0, c, len = strlen(s);
9c94      rep(i, len) {
3140        c = id(s[i]);
f339        if (!tr[p][c]) return 0;
f119        p = tr[p][c];
95cf      }
840e      return tag[p];
95cf    }
329b  };
```

3.5 Rolling hash

PLEASE call `init_hash()` in `int main()`!

Usage:

`build(str)`	Construct the hasher with given string.
`operator()(l, r)`	Get hash value of substring $[l, r)$.

```
1e42  const LL mod = 1006658951440146419, g = 967;
9f60  const int MAXN = 200005;
0291  LL pg[MAXN];
427e
6832  inline LL mul(LL x, LL y) {
c919      return __int128_t(x) * y % mod;
95cf  }
427e
599a  void init_hash() {   // must be called in `int main()`
286f      pg[0] = 1;
d00f      for (int i = 1; i < MAXN; i++)
4aa9          pg[i] = pg[i - 1] * g % mod;
95cf  }
```

```
427e
7e62 struct hasher {
534a     LL val[MAXN];
427e
4554     void build(const char *str) {   // assume lower-case letter only
f937         for (int i = 0; str[i]; i++)
9645             val[i+1] = (mul(val[i], g) + str[i]) % mod;
95cf     }
427e
19f8     LL operator() (int l, int r) { // [l, r)
9986         return (val[r] - mul(val[l], pg[r - l]) + mod) % mod;
95cf     }
b179 } ha;
```

4 Math

4.1 Matrix powermod

```
44b4 const int MAXN = 105;
92df const LL modular = 1000000007;
5c83 int n; // order of matrices
427e
8864 struct matrix{
3180     LL m[MAXN][MAXN];
427e
43c5     void operator *=(matrix& a){
e735         static LL t[MAXN][MAXN];
34d7         Rep (i, n){
4c11             Rep (j, n){
ee1e                 t[i][j] = 0;
c4a7                 Rep (k, n){
fcaf                     t[i][j] += (m[i][k] * a.m[k][j]) % modular;
199e                     t[i][j] %= modular;
95cf                 }
95cf             }
95cf         }
dad4         memcpy(m, t, sizeof(t));
95cf     }
329b };
427e
63d8 matrix r;
3ec2 void m_powmod(matrix& b, LL e){
83f0     memset(r.m, 0, sizeof(r.m));
a7c3     Rep(i, n)
de64         r.m[i][i] = 1;
3e90     while (e){
5a0e         if (e & 1) r *= b;
35c5         b *= b;
16fc         e >>= 1;
95cf     }
95cf }
```

4.2 Linear basis

```
8b44 const int MAXD = 30;
03a6 struct linearbasis {
3558     ULL b[MAXD] = {};
427e
842f     bool insert(ll v) {
9b2b         for (int j = MAXD - 1; j >= 0; j--) {
de36             if (!(v & (1ll << j))) continue;
ee78             if (b[j]) v ^= b[j]
037f             else {
7836                 for (int k = 0; k < j; k++)
f0b4                     if (v & (1ll << k)) v ^= b[k];
b0aa                 for (int k = j + 1; k < MAXD; k++)
46c9                     if (b[k] & (1ll << j)) b[k] ^= v;
8295                 b[j] = v;
3361                 return true;
95cf             }
95cf         }
438e         return false;
95cf     }
329b };
```

4.3 Gauss elimination over finite field

```
b784 const LL p = 1000000007;
427e
2a2c LL powmod(LL b, LL e) {
95a2   LL r = 1;
3e90   while (e) {
```

```
1783    if (e & 1) r = r * b % p;
5549    b = b * b % p;
16fc    e >>= 1;
95cf  }
547e  return r;
95cf }
427e
c130 typedef vector<LL> VLL;
42ac typedef vector<VLL> VVLL;
427e
2c62 LL gauss(VVLL &a, VVLL &b) {
561b  const int n = a.size(), m = b[0].size();
a25e  vector<int> irow(n), icol(n), ipiv(n);
2976  LL det = 1;
427e
be8e  rep (i, n) {
d2b5    int pj = -1, pk = -1;
6b4a    rep (j, n) if (!ipiv[j])
e582      rep (k, n) if (!ipiv[k])
6112        if (pj == -1 || a[j][k] > a[pj][pk]) {
a905          pj = j;
657b          pk = k;
95cf        }
d480    if (a[pj][pk] == 0) return 0;
0305    ipiv[pk]++;
8dad    swap(a[pj], a[pk]);
aad8    swap(b[pj], b[pk]);
be4d    if (pj != pk) det = (p - det) % p;
d080    irow[i] = pj;
f156    icol[i] = pk;
427e
4ecd    LL c = powmod(a[pk][pk], p - 2);
865b    det = det * a[pk][pk] % p;
c36a    a[pk][pk] = 1;
dd36    rep (j, n) a[pk][j] = a[pk][j] * c % p;
1b23    rep (j, m) b[pk][j] = b[pk][j] * c % p;
f8f3    rep (j, n) if (j != pk) {
e97f      c = a[j][pk];
c449      a[j][pk] = 0;
820b      rep (k, n) a[j][k] = (a[j][k] + p - a[pk][k] * c % p) % p;
f039      rep (k, m) b[j][k] = (b[j][k] + p - b[pk][k] * c % p) % p;
95cf    }
95cf  }
427e
37e1  for (int j = n - 1; j >= 0; j--) if (irow[j] != icol[j]) {
50dc    for (int k = 0; k < n; k++) swap(a[k][irow[j]], a[k][icol[j]]);
95cf  }
f27f  return det;
95cf }
```

4.4 Berlekamp-Massey algorithm

```
2b86 const LL MOD = 1000000007;
427e
391d LL inverse(LL b) {
32d3  LL e = MOD - 2, r = 1;
3e90  while (e) {
9a62    if (e & 1) r = r * b % MOD;
29ea    b = b * b % MOD;
16fc    e >>= 1;
95cf  }
547e  return r;
95cf }
427e
32a6 struct Poly {
afe0  vector<int> a;
427e
9794  Poly() { a.clear(); }
427e
de81  Poly(vector<int> &a) : a(a) {}
427e
8087  int length() const { return a.size(); }
427e
16de  Poly move(int d) {
b31d    vector<int> na(d, 0);
f915    na.insert(na.end(), a.begin(), a.end());
cecf    return Poly(na);
95cf  }
427e
fa1a  int calc(vector<int> &d, int pos) {
5b57    int ret = 0;
501c    for (int i = 0; i < (int)a.size(); ++i) {
5de5      if ((ret += (long long)d[pos - i] * a[i] % MOD) >= MOD) {
3041        ret -= MOD;
95cf      }
95cf    }
```

```
ee0f     return ret;
95cf   }
427e
c856   Poly operator - (const Poly &b) {
bd55     vector<int> na(max(this->length(), b.length()));
d1a7     for (int i = 0; i < (int)na.size(); ++i) {
3507       int aa = i < this->length() ? this->a[i] : 0,
2bee           bb = i < b.length() ? b.a[i] : 0;
9526       na[i] = (aa + MOD - bb) % MOD;
95cf     }
cecf     return Poly(na);
95cf   }
329b };
427e
5473 Poly operator * (const int &c, const Poly &p) {
72de   vector<int> na(p.length());
d1a7   for (int i = 0; i < (int)na.size(); ++i) {
bf0c     na[i] = (long long)c * p.a[i] % MOD;
95cf   }
aaab   return na;
95cf }
427e
afff vector<int> solve(vector<int> a) {
9f23   int n = a.size();
58d0   Poly s, b;
4e8f   s.a.push_back(1), b.a.push_back(1);
c2aa   for (int i = 1, j = 0, ld = a[0]; i < n; ++i) {
4158     int d = s.calc(a, i);
d503     if (d) {
c29d       if ((s.length() - 1) * 2 <= i) {
db9d         Poly ob = b;
6bce         b = s;
1d0e         s = s - (long long)d * inverse(ld) % MOD * ob.move(i - j);
0889         j = i;
64f1         ld = d;
8e2e       } else {
714e         s = s - (long long)d * inverse(ld) % MOD * b.move(i - j);
95cf       }
95cf     }
95cf   }
427e   // Caution: s.a might be shorter than expected
e235   return s.a;
95cf }
```

4.5 Fast Walsh-Hadamard transform

```
061e void fwt(int* a, int n){
5595     for (int d = 1; d < n; d <<= 1)
05f2         for (int i = 0; i < n; i += d << 1)
b833             rep (j, d){
7796                 int x = a[i+j], y = a[i+j+d];
427e                 // a[i+j] = x+y, a[i+j+d] = x-y;    // xor
427e                 // a[i+j] = x+y;                    // and
427e                 // a[i+j+d] = x+y;                  // or
95cf             }
95cf }
427e
4db1 void ifwt(int* a, int n){
5595     for (int d = 1; d < n; d <<= 1)
05f2         for (int i = 0; i < n; i += d << 1)
b833             rep (j, d){
7796                 int x = a[i+j], y = a[i+j+d];
427e                 // a[i+j] = (x+y)/2, a[i+j+d] = (x-y)/2;    // xor
427e                 // a[i+j] = x-y;                            // and
427e                 // a[i+j+d] = y-x;                          // or
95cf             }
95cf }
427e
2ab6 void conv(int* a, int* b, int n){
950a     fwt(a, n);
e427     fwt(b, n);
8a42     rep(i, n) a[i] *= b[i];
430f     ifwt(a, n);
95cf }
```

4.6 Fast fourier transform

```
4e09 const int NMAX = 1<<20;
427e
3fbf typedef complex<double> cplx;
427e
abd1 const double PI = 2*acos(0.0);
12af struct FFT{
c47c     int rev[NMAX];
27d7     cplx omega[NMAX], oinv[NMAX];
9827     int K, N;
```

```
427e
1442      FFT(int k){
e209          K = k; N = 1 << k;
b393          rep (i, N){
7ba3              rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
1908              omega[i] = polar(1.0, 2.0 * PI / N * i);
a166              oinv[i] = conj(omega[i]);
95cf          }
95cf      }
427e
b941      void dft(cplx* a, cplx* w){
a215          rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
ac6e          for (int l = 2; l <= N; l *= 2){
2969              int m = l/2;
b3cf              for (cplx* p = a; p != a + N; p += l)
c24f                  rep (k, m){
fe06                      cplx t = w[N/l*k] * p[k+m];
ecbf                      p[k+m] = p[k] - t; p[k] += t;
95cf                  }
95cf          }
95cf      }
427e
617b      void fft(cplx* a){dft(a, omega);}
a123      void ifft(cplx* a){
3b2f          dft(a, oinv);
57fc          rep (i, N) a[i] /= N;
95cf      }
427e
bdc0      void conv(cplx* a, cplx* b){
6497          fft(a); fft(b);
12a5          rep (i, N) a[i] *= b[i];
f84e          ifft(a);
95cf      }
329b  };
```

4.7 Number theoretic transform

```
4ab9  const int NMAX = 1<<21;
427e
427e  // 998244353 = 7*17*2^23+1, G = 3
fb9a  const int P = 1004535809, G = 3; // = 479*2^21+1
427e
```

```
87ab  struct NTT{
c47c      int rev[NMAX];
0eda      LL omega[NMAX], oinv[NMAX];
81af      int g, g_inv; // g: g_n = G^((P-1)/n)
9827      int K, N;
427e
2a2c      LL powmod(LL b, LL e){
95a2          LL r = 1;
3e90          while (e){
6624              if (e&1) r = r * b % P;
489e              b = b * b % P;
16fc              e >>= 1;
95cf          }
547e          return r;
95cf      }
427e
f420      NTT(int k){
e209          K = k; N = 1 << k;
7652          g = powmod(G, (P-1)/N);
4b3a          g_inv = powmod(g, N-1);
e04f          omega[0] = oinv[0] = 1;
b393          rep (i, N){
7ba3              rev[i] = (rev[i>>1]>>1) | ((i&1)<<(K-1));
ad4f              if (i){
8d8b                  omega[i] = omega[i-1] * g % P;
9e14                  oinv[i] = oinv[i-1] * g_inv % P;
95cf              }
95cf          }
95cf      }
427e
9668      void _ntt(LL* a, LL* w){
a215          rep (i, N) if (i < rev[i]) swap(a[i], a[rev[i]]);
ac6e          for (int l = 2; l <= N; l *= 2){
2969              int m = l/2;
7a1d              for (LL* p = a; p != a + N; p += l)
c24f                  rep (k, m){
0ad3                      LL t = w[N/l*k] * p[k+m] % P;
6209                      p[k+m] = (p[k] - t + P) % P;
fa1b                      p[k] = (p[k] + t) % P;
95cf                  }
95cf          }
95cf      }
427e
92ea      void ntt(LL* a){_ntt(a, omega);}
```

```
5daf     void intt(LL* a){
1f2a         LL inv = powmod(N, P-2);
9910         _ntt(a, oinv);
a873         rep (i, N) a[i] = a[i] * inv % P;
95cf     }
427e
3a5b     void conv(LL* a, LL* b){
ad16         ntt(a); ntt(b);
e49e         rep (i, N) a[i] = a[i] * b[i] % P;
5748         intt(a);
95cf     }
329b };
```

4.8 Sieve of Euler

```
cfc3 const int MAXX = 1e7+5;
5861 bool p[MAXX];
73ae int prime[MAXX], sz;
427e
9bc6 void sieve(){
9628     p[0] = p[1] = 1;
1ec8     for (int i = 2; i < MAXX; i++){
bf28         if (!p[i]) prime[sz++] = i;
e82c         for (int j = 0; j < sz && i*prime[j] < MAXX; j++){
b6a9             p[i*prime[j]] = 1;
5f51             if (i % prime[j] == 0) break;
95cf         }
95cf     }
95cf }
```

4.9 Sieve of Euler (General)

```
b62e namespace sieve {
6589   constexpr int MAXN = 10000007;
e982   bool p[MAXN]; // true if not prime
6ae8   int prime[MAXN], sz;
cbf7   int pval[MAXN], pcnt[MAXN];
6030   int f[MAXN];
427e
76f6   void exec(int N = MAXN) {
9628     p[0] = p[1] = 1;
427e     pval[1] = 1;
8a8a     pcnt[1] = 0;
bdda     f[1] = 1;
c6b9
427e     for (int i = 2; i < N; i++) {
a643       if (!p[i]) {
01d6         prime[sz++] = i;
b2b2         for (LL j = i; j < N; j *= i) {
37d9           int b = j / i;
758c           pval[j] = i * pval[b];
81fd           pcnt[j] = pcnt[b] + 1;
e0f3           f[j] = ______; // f[j] = f(i^pcnt[j])
a96c         }
95cf       }
95cf       for (int j = 0; i * prime[j] < N; j++) {
34c0         int x = i * prime[j]; p[x] = 1;
f87a         if (i % prime[j] == 0) {
20cc           pval[x] = pval[i] * prime[j];
9985           pcnt[x] = pcnt[i] + 1;
3f93         } else {
8e2e           pval[x] = prime[j];
cc91           pcnt[x] = 1;
6322         }
95cf         if (x != pval[x]) {
6191           f[x] = f[x / pval[x]] * f[pval[x]]
d614         }
95cf         if (i % prime[j] == 0) break;
5f51       }
95cf     }
95cf   }
95cf }
```

4.10 Miller-Rabin primality test

The array `a[]` (excluding senitel, i.e. `LLONG_MAX`) should be

`{2}`	when $n < 2,047$.
`{2, 7, 61}`	when $n < 4,759,123,141$ (2^{32}).
`{2, 3, 5, 7, 11}`	when $n < 2.1 \times 10^{12}$.
`{2, 325, 9375, 28178, 450775, 9780504, 1795265022}`	when $n < 2^{64}$.

```
f16f bool test(LL n){
```

```
59f2     if (n < 3) return n==2;
427e     // ! The array a[] should be modified if the range of x changes.
3f11     const LL a[] = {2LL, 7LL, 61LL, LLONG_MAX};
c320     LL r = 0, d = n-1, x;
f410     while (~d & 1) d >>= 1, r++;
2975     for (int i=0; a[i] < n; i++){
ece1         x = powmod(a[i], d, n); // ! powmod must use for 64bit mulmod
7f99         if (x == 1 || x == n-1) goto next;
e257         rep (i, r) {
d7ff             x = mulmod(x, x, n);
8d2e             if (x == n-1) goto next;
95cf         }
438e         return false;
d490 next:;
95cf     }
3361     return true;
95cf }
```

4.11 Pollard's rho algorithm

```
2e6b ULL gcd(ULL a, ULL b) {return b ? gcd(b, a % b) : a;}
427e
54a5 ULL PollardRho(ULL n){
45eb     ULL c, x, y, d = n;
d3e5     if (~n&1) return 2;
3c69     while (d == n){
0964         x = y = 2;
4753         d = 1;
5952         c = rand() % (n - 1) + 1;
9e5b         while (d == 1){
33d5             x = (mulmod(x, x, n) + c) % n;
e1bf             y = (mulmod(y, y, n) + c) % n;
e1bf             y = (mulmod(y, y, n) + c) % n;
a313             d = gcd(x>y ? x-y : y-x, n);
95cf         }
95cf     }
5d89     return d;
95cf }
```

4.12 Qusai-polynomial sum

Must call `init()` before use!

```
b24e namespace polysum {
1dc8 #define rep(i, a, n) for (int i = a; i < n; i++)
1481 #define per(i, a, n) for (int i = n - 1; i >= a; i--)
3946 const int D = 2010;
c076 ll a[D], f[D], g[D], p[D], p1[D], p2[D], b[D], h[D][2], C[D];
c4cb ll powmod(ll a, ll b) {
e4b7   ll res = 1;
af5c   a %= mod;
6e39   assert(b >= 0);
b1fa   for (; b; b >>= 1) {
0684     if (b & 1) res = res * a % mod;
05a8     a = a * a % mod;
95cf   }
244d   return res;
95cf }
e88b ll calcn(int d, ll *a, ll n) {  // a[0].. a[d]  a[n]
b4aa   if (n <= d) return a[n];
d6be   p1[0] = p2[0] = 1;
3245   rep(i, 0, d + 1) {
ffec     ll t = (n - i + mod) % mod;
532d     p1[i + 1] = p1[i] * t % mod;
95cf   }
3245   rep(i, 0, d + 1) {
9800     ll t = (n - d + i + mod) % mod;
9f60     p2[i + 1] = p2[i] * t % mod;
95cf   }
19f3   ll ans = 0;
3245   rep(i, 0, d + 1) {
860e     ll t = g[i] * g[d - i] % mod * p1[i] % mod * p2[d - i] % mod * a[i] % mod;
752a     if ((d - i) & 1)
a69f       ans = (ans - t + mod) % mod;
649a     else
29fe       ans = (ans + t) % mod;
95cf   }
4206   return ans;
95cf }
1901 void init(int M) {
6323   f[0] = f[1] = g[0] = g[1] = 1;
fe69   rep(i, 2, M + 5) f[i] = f[i - 1] * i % mod;
b375   g[M + 4] = powmod(f[M + 4], mod - 2);
```

```
7e87   per(i, 1, M + 4) g[i] = g[i + 1] * (i + 1) % mod;
95cf }
5f6d ll polysum(ll m, ll *a, ll n) {  // a[0].. a[m] \sum_{i=0}^{n-1} a[i]
2f0c   ll b[D];
a950   for (int i = 0; i <= m; i++) b[i] = a[i];
96b8   b[m + 1] = calcn(m, b, m + 1);
7785   rep(i, 1, m + 2) b[i] = (b[i - 1] + b[i]) % mod;
cc07   return calcn(m + 1, b, n - 1);
95cf }
c704 ll qpolysum(ll R, ll n, ll *a, ll m) {  // a[0].. a[m] \sum_{i=0}^{n-1} a[i]*R^i
356d   if (R == 1) return polysum(n, a, m);
ee67   a[m + 1] = calcn(m, a, m + 1);
2f7b   ll r = powmod(R, mod - 2), p3 = 0, p4 = 0, c, ans;
c222   h[0][0] = 0;
c576   h[0][1] = 1;
4d99   rep(i, 1, m + 2) {
dcbd     h[i][0] = (h[i - 1][0] + a[i - 1]) * r % mod;
3f1a     h[i][1] = h[i - 1][1] * r % mod;
95cf   }
dc94   rep(i, 0, m + 2) {
2d72     ll t = g[i] * g[m + 1 - i] % mod;
59aa     if (i & 1)
60b1       p3 = ((p3 - h[i][0] * t) % mod + mod) % mod,
19f7       p4 = ((p4 - h[i][1] * t) % mod + mod) % mod;
649a     else
b9ee       p3 = (p3 + h[i][0] * t) % mod, p4 = (p4 + h[i][1] * t) % mod;
95cf   }
6eed   c = powmod(p4, mod - 2) * (mod - p3) % mod;
a893   rep(i, 0, m + 2) h[i][0] = (h[i][0] + h[i][1] * c) % mod;
9267   rep(i, 0, m + 2) C[i] = h[i][0];
8a10   ans = (calcn(m, C, n) * powmod(R, n) - c) % mod;
2dc8   if (ans < 0) ans += mod;
4206   return ans;
95cf }
95cf } // namespace polysum
```

5 Graph Theory

5.1 Strongly connected component

```
837c const int MAXV = 100005;

struct graph{
    vector<int> adj[MAXV];
    stack<int> s;
    int V; // number of vertices
    int pre[MAXV], lnk[MAXV], scc[MAXV];
    int time, sccn;

    void add_edge(int u, int v){
        adj[u].push_back(v);
    }

    void dfs(int u){
        pre[u] = lnk[u] = ++time;
        s.push(u);
        for (int v : adj[u]){
            if (!pre[v]){
                dfs(v);
                lnk[u] = min(lnk[u], lnk[v]);
            } else if (!scc[v]){
                lnk[u] = min(lnk[u], pre[v]);
            }
        }
        if (lnk[u] == pre[u]){
            sccn++;
            int x;
            do {
                x = s.top(); s.pop();
                scc[x] = sccn;
            } while (x != u);
        }
    }

    void find_scc(){
        time = sccn = 0;
        memset(scc, 0, sizeof scc);
        memset(pre, 0, sizeof pre);
        Rep (i, V){
            if (!pre[i]) dfs(i);
        }
    }

    vector<int> adjc[MAXV];
    void contract(){
```

```
        Rep (i, V)
            rep (j, adj[i].size()){
                if (scc[i] != scc[adj[i][j]])
                    adjc[scc[i]].push_back(scc[adj[i][j]]);
            }
    }
};
```

5.2 Vertex biconnected component

```
const int MAXN = 100005;
struct graph {
    int pre[MAXN], iscut[MAXN], bccno[MAXN], dfs_clock, bcc_cnt;
    vector<int> adj[MAXN], bcc[MAXN];
    set<pair<int, int>> bcce[MAXN];

    stack<pair<int, int>> s;

    void add_edge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    int dfs(int u, int fa) {
        int lowu = pre[u] = ++dfs_clock;
        int child = 0;
        for (int v : adj[u]) {
            if (!pre[v]) {
                s.push({u, v});
                child++;
                int lowv = dfs(v, u);
                lowu = min(lowu, lowv);
                if (lowv >= pre[u]) {
                    iscut[u] = 1;
                    bcc[bcc_cnt].clear();
                    bcce[bcc_cnt].clear();
                    while (1) {
                        int xu, xv;
                        tie(xu, xv) = s.top(); s.pop();
                        bcce[bcc_cnt].insert({min(xu, xv), max(xu, xv)});
                        if (bccno[xu] != bcc_cnt) {
                            bcc[bcc_cnt].push_back(xu);
                            bccno[xu] = bcc_cnt;
                        }
                        if (bccno[xv] != bcc_cnt) {
                            bcc[bcc_cnt].push_back(xv);
                            bccno[xv] = bcc_cnt;
                        }
                        if (xu == u && xv == v) break;
                    }
                    bcc_cnt++;
                }
            } else if (pre[v] < pre[u] && v != fa) {
                s.push({u, v});
                lowu = min(lowu, pre[v]);
            }
        }
        if (fa < 0 && child == 1) iscut[u] = 0;
        return lowu;
    }

    void find_bcc(int n) {
        memset(pre, 0, sizeof pre);
        memset(iscut, 0, sizeof iscut);
        memset(bccno, -1, sizeof bccno);
        dfs_clock = bcc_cnt = 0;
        rep (i, n) if (!pre[i]) dfs(i, -1);
    }
};
```

5.3 Minimum spanning arborescence (Chu-Liu)

All vertices are 1-based.

Usage:

`getans(n, root, edges)`	Compute the total size of MSA rooted at `root`.

Time Complexity: $O(|V||E|)$

```
struct edge {
    int u, v;
    LL w;
};

const int MAXN = 10005;
LL in[MAXN];
```

```
1c1d int pre[MAXN], vis[MAXN], id[MAXN];
427e
5a43 LL getans(int n, int rt, vector<edge>& edges) {
f7ff     LL ans = 0;
8abb     int cnt = 0;
a147     while (1) {
641a         Rep (i, n) in[i] = LLONG_MAX, id[i] = vis[i] = 0;
0705         for (auto e : edges) {
073a             if (e.u != e.v and e.w < in[e.v]) {
c1df                 pre[e.v] = e.u;
5fbc                 in[e.v] = e.w;
95cf             }
95cf         }
3fdb         in[rt] = 0;
34d7         Rep (i, n) {
3c97             if (in[i] == LLONG_MAX) return -1;
cf57             ans += in[i];
a763             int u;
4b0e             for (u = i; u != rt && vis[u] != i && !id[u]; u = pre[u])
88a2                 vis[u] = i;
4b22             if (u != rt && !id[u]) {
b66e                 id[u] = ++cnt;
0443                 for (int v = pre[u]; v != u; v = pre[v])
5c22                     id[v] = cnt;
95cf             }
95cf         }
91e9         if (!cnt) return ans;
5e22         Rep (i, n) if (!id[i]) id[i] = ++cnt;
7400         for (auto& e : edges) {
7750             LL laz = in[e.v];
97ae             e.u = id[e.u];
fae6             e.v = id[e.v];
bdd2             if (e.u != e.v) e.w -= laz;
95cf         }
6cc4         n = cnt; rt = id[rt]; cnt = 0;
95cf     }
95cf }
```

5.4 Maximum flow (Dinic)

Usage:

`add_edge(u, v, c)`	Add an edge from u to v with capacity c.
`max_flow(s, t)`	Compute maximum flow from s to t.

Time Complexity: For general graph, $O(V^2E)$; for network with unit capacity, $O(\min\{V^{2/3}, \sqrt{E}\}E)$; for bipartite network, $O(\sqrt{V}E)$.

```
bcf8 struct edge{
60e2     int from, to;
5e6d     LL cap, flow;
329b };
427e
e2cd const int MAXN = 1005;
9062 struct Dinic {
4dbf     int n, m, s, t;
9f0c     vector<edge> edges;
b891     vector<int> G[MAXN];
bbb6     bool vis[MAXN];
b40a     int d[MAXN];
ddec     int cur[MAXN];
427e
5973     void add_edge(int from, int to, LL cap) {
7b55         edges.push_back(edge{from, to, cap, 0});
1db7         edges.push_back(edge{to, from, 0, 0});
fe77         m = edges.size();
dff5         G[from].push_back(m-2);
8f2d         G[to].push_back(m-1);
95cf     }
427e
1836     bool bfs() {
3b73         memset(vis, 0, sizeof(vis));
93d2         queue<int> q;
5d13         q.push(s);
2cd2         vis[s] = 1;
721d         d[s] = 0;
cc78         while (!q.empty()) {
66ba             int x = q.front(); q.pop();
3b61             for (int i = 0; i < G[x].size(); i++) {
b510                 edge& e = edges[G[x][i]];
bba9                 if (!vis[e.to] && e.cap > e.flow) {
cd72                     vis[e.to] = 1;
cf26                     d[e.to] = d[x] + 1;
ca93                     q.push(e.to);
95cf                 }
95cf             }
95cf         }
```

```
b23b        return vis[t];
95cf    }
427e
9252    LL dfs(int x, LL a) {
6904        if (x == t || a == 0) return a;
8bf9        LL flow = 0, f;
f515        for (int& i = cur[x]; i < G[x].size(); i++) {
b510            edge& e = edges[G[x][i]];
2374            if(d[x] + 1 == d[e.to] && (f = dfs(e.to, min(a, e.cap-e.flow))) > 0)
                    {
1cce                e.flow += f;
e16d                edges[G[x][i]^1].flow -= f;
a74d                flow += f;
23e5                a -= f;
97ed                if(a == 0) break;
95cf            }
95cf        }
84fb        return flow;
95cf    }
427e
5bf2    LL max_flow(int s, int t) {
590d        this->s = s; this->t = t;
62e2        LL flow = 0;
ed58        while (bfs()) {
f326            memset(cur, 0, sizeof(cur));
fb3a            flow += dfs(s, LLONG_MAX);
95cf        }
84fb        return flow;
95cf    }
427e
c72e    vector<int> min_cut() { // call this after maxflow
1df9        vector<int> ans;
df9a        for (int i = 0; i < edges.size(); i++) {
56d8            edge& e = edges[i];
46a2            if(vis[e.from] && !vis[e.to] && e.cap > 0) ans.push_back(i);
95cf        }
4206        return ans;
95cf    }
329b};
```

5.5 Maximum cardinality bipartite matching (Hungarian)

```
302f#include <bits/stdc++.h>
421cusing namespace std;
427e
0d6c#define rep(i, n) for (int i = 0; i < (n); i++)
cfe3#define Rep(i, n) for (int i = 1; i <= (n); i++)
8843#define range(x) (x).begin(), (x).end()
5cadtypedef long long LL;
427e
84eestruct Hungarian{
fbf6    int nx, ny;
9ec6    vector<int> mx, my;
9d4c    vector<vector<int> > e;
edec    vector<bool> mark;
427e
8324    void init(int nx, int ny){
c1d1        this->nx = nx;
f9c1        this->ny = ny;
ac92        mx.resize(nx); my.resize(ny);
3f11        e.clear(); e.resize(nx);
1023        mark.resize(nx);
95cf    }
427e
4589    inline void add(int a, int b){
486c        e[a].push_back(b);
95cf    }
427e
0c2b    bool augment(int i){
207c        if (!mark[i]) {
dae4            mark[i] = true;
6a1e            for (int j : e[i]){
0892                if (my[j] == -1 || augment(my[j])){
9ca3                    mx[i] = j; my[j] = i;
3361                    return true;
95cf                }
95cf            }
95cf        }
438e        return false;
95cf    }
427e
3fac    int match(){
5b57        int ret = 0;
b0f1        fill(range(mx), -1);
b957        fill(range(my), -1);
4ed1        rep (i, nx){
```

```
13a5            fill(range(mark), false);
cc89            if (augment(i)) ret++;
95cf        }
ee0f        return ret;
95cf    }
329b };
```

5.6 Minimum cost maximum flow

```
bcf8 struct edge{
60e2     int from, to;
d698     int cap, flow;
32cc     LL cost;
329b };
427e
cc3e const LL INF = LLONG_MAX / 2;
2aa8 const int MAXN = 5005;
c6cb struct MCMF {
9ceb     int s, t, n, m;
9f0c     vector<edge> edges;
b891     vector<int> G[MAXN];
f74f     bool inq[MAXN]; // queue
8f67     LL d[MAXN];     // distance
9524     int p[MAXN];    // previous
b330     int a[MAXN];    // improvement
427e
f7f2     void add_edge(int from, int to, int cap, LL cost) {
24f0         edges.push_back(edge{from, to, cap, 0, cost});
95f0         edges.push_back(edge{to, from, 0, 0, -cost});
fe77         m = edges.size();
dff5         G[from].push_back(m-2);
8f2d         G[to].push_back(m-1);
95cf     }
427e
3c52     bool spfa(){
93d2         queue<int> q;
8494         fill(d, d + MAXN, INF); d[s] = 0;
fd48         memset(inq, 0, sizeof(inq));
5e7c         q.push(s); inq[s] = true;
2dae         p[s] = 0; a[s] = INT_MAX;
cc78         while (!q.empty()){
b0aa             int u = q.front(); q.pop(); inq[u] = false;
3bba             for (int i : G[u]) {
56d8                 edge& e = edges[i];
3601                 if (e.cap > e.flow && d[e.to] > d[u] + e.cost){
55bc                     d[e.to] = d[u] + e.cost;
0bea                     p[e.to] = G[u][i];
8249                     a[e.to] = min(a[u], e.cap - e.flow);
e5d3                     if (!inq[e.to]) q.push(e.to), inq[e.to] = true;
95cf                 }
95cf             }
95cf         }
6d7c         return d[t] != INF;
95cf     }
427e
71a4     void augment(){
06f1         int u = t;
b19d         while (u != s){
db09             edges[p[u]].flow += a[t];
25a9             edges[p[u]^1].flow -= a[t];
e6c9             u = edges[p[u]].from;
95cf         }
95cf     }
427e
6e20 #ifdef GIVEN_FLOW
5972     bool min_cost(int s, int t, int f, LL& cost) {
590d         this->s = s; this->t = t;
21d4         int flow = 0;
23cb         cost = 0;
22dc         while (spfa()) {
bcdb             augment();
a671             if (flow + a[t] >= f){
b14d                 cost += (f - flow) * d[t]; flow = f;
3361                 return true;
8e2e             } else {
2a83                 flow += a[t]; cost += a[t] * d[t];
95cf             }
95cf         }
438e         return false;
95cf     }
a8cb #else
f9a9     int min_cost(int s, int t, LL& cost) {
590d         this->s = s; this->t = t;
21d4         int flow = 0;
23cb         cost = 0;
22dc         while (spfa()) {
```

```
            augment();
            flow += a[t]; cost += a[t] * d[t];
        }
        return flow;
    }
#endif
};
```

5.7 Global minimum cut (Stoer-Wagner)

```
typedef vector<LL> VI;
typedef vector<VI> VVI;

pair<LL, VI> stoer(VVI &w) {
    int n = w.size();
    VI used(n), c, bestc;
    LL bestw = -1;

    for (int ph = n - 1; ph >= 0; ph--) {
        VI wt = w[0], added = used;
        int prev, last = 0;
        rep (i, ph) {
            prev = last;
            last = -1;
            for (int j = 1; j < n; j++)
                if (!added[j] && (last == -1 || wt[j] > wt[last]))
                    last = j;
            if (i == ph - 1) {
                rep (j, n) w[prev][j] += w[last][j];
                rep (j, n) w[j][prev] = w[prev][j];
                used[last] = true;
                c.push_back(last);
                if (bestw == -1 || wt[last] < bestw) {
                    bestc = c;
                    bestw = wt[last];
                }
            } else {
                rep (j, n) wt[j] += w[last][j];
                added[last] = true;
            }
        }
    }
    return {bestw, bestc};
}
```

5.8 Heavy-light decomposition

Time Complexity: The decomposition itself takes linear time. Each query takes $O(\log n)$ operations.

```
const int MAXN = 100005;
vector<int> adj[MAXN];
int sz[MAXN], top[MAXN], fa[MAXN], son[MAXN], depth[MAXN], id[MAXN];

void dfs1(int x, int dep, int par){
    depth[x] = dep;
    sz[x] = 1;
    fa[x] = par;
    int maxn = 0, s = 0;
    for (int c: adj[x]){
        if (c == par) continue;
        dfs1(c, dep + 1, x);
        sz[x] += sz[c];
        if (sz[c] > maxn){
            maxn = sz[c];
            s = c;
        }
    }
    son[x] = s;
}

int cid = 0;
void dfs2(int x, int t){
    top[x] = t;
    id[x] = ++cid;
    if (son[x]) dfs2(son[x], t);
    for (int c: adj[x]){
        if (c == fa[x]) continue;
        if (c == son[x]) continue;
        else dfs2(c, c);
    }
}

void decomp(int root){
    dfs1(root, 1, 0);
```

```
1c88     dfs2(root, root);
95cf }
427e
2c98 void query(int u, int v){
03a1     while (top[u] != top[v]){
45ec         if (depth[top[u]] < depth[top[v]]) swap(u, v);
427e         // id[top[u]] to id[u]
005b         u = fa[top[u]];
95cf     }
6083     if (depth[u] > depth[v]) swap(u, v);
427e     // id[u] to id[v]
95cf }
```

5.9 DSU on tree

DSU on tree. This avoids parallel existence of multiple data structures but requires that the data structure is invertible. To use this template, implement `merge`, `enter`, `leave` as needed; first call `decomp(root, 0)`, then call `work(root, 0, false)`. Labels of vertices start from 1.

Usage:

`decomp(u, p)`	Decompose the tree u.
`work(u, p, keep)`	Work for subtree u. When `keep` is set, information is not cleared.

Time Complexity: $O(n \log n)$ times the complexity for `merge`, `enter`, `leave`.

```
1fb6 vector<int> adj[100005];
901d int sz[100005], son[100005];
427e
5559 void decomp(int u, int p) {
50c0     sz[u] = 1;
18f6     for (int v : adj[u]) {
bd87         if (v == p) continue;
a851         decomp(v, u);
8449         sz[u] += sz[v];
d28c         if (sz[v] > sz[son[u]]) son[u] = v;
95cf     }
95cf }
427e
b7ec template <typename T>
62f5 void trav(T fn, int u, int p) {
4412     fn(u);
30b3     for (int v : adj[u]) if (v != p) trav(fn, v, u);
95cf }
427e
7467 #define for_light(v) for (int v : adj[u]) if (v != p and v != son[u])
33ff void work(int u, int p, bool keep) {
72a2     for_light(v) work(v, u, 0); // process light children
427e
427e     // process heavy child
427e     // current data structure contains info of heavy child
9866     if (son[u]) work(son[u], u, 1);
427e
18a9     auto merge = [u] (int c) { /* count contribution of c */ };
1ab0     auto enter = [] (int c) { /* add vertex c */ };
f241     auto leave = [] (int c) { /* remove vertex c*/ };
427e
3d3b     for_light(v) {
74c6         trav(merge, v, u);
c13d         trav(enter, v, u);
95cf     }
427e
427e     // count answer for root and add it
427e     // Warning: special check may apply to root!
c54f     merge(u);
9dec     enter(u);
427e
427e     // leave current tree
4e3e     if (!keep) trav(leave, u, p);
95cf }
```

6 Data Structures

6.1 Fenwick tree (point update range query)

```
9976 struct bit_purq { // point update, range query
d7af     int N;
99ff     vector<LL> tr;
427e
d34f     void init(int n) { // fill the array with 0
1010         tr.resize(N = n + 5);
95cf     }
427e
63d0     LL sum(int n) {
```

```
f7ff        LL ans = 0;
e290        while (n) {
0715            ans += tr[n];
c0d4            n &= n - 1;
95cf        }
4206        return ans;
95cf    }
427e
f4bd    void add(int n, LL x){
ad20        while (n < N) {
6c81            tr[n] += x;
0af5            n += n & -n;
95cf        }
95cf    }
329b };
```

6.2 Fenwick tree (range update point query)

```
3d03 struct bit_rupq{ // range update, point query
d7af     int N;
99ff     vector<LL> tr;
427e
d34f     void init(int n) { // fill the array with 0
1010         tr.resize(N = n + 5);
95cf     }
427e
38d4     LL query(int n) {
f7ff         LL ans = 0;
ad20         while (n < N) {
0715             ans += tr[n];
0af5             n += n & -n;
95cf         }
4206         return ans;
95cf     }
427e
f4bd     void add(int n, LL x) {
e290         while (n){
6c81             tr[n] += x;
c0d4             n &= n - 1;
95cf         }
95cf     }
329b };
```

6.3 Segment tree

```
LL p;                                                                   3942
const int MAXN = 4 * 100006;                                            1ebb
struct segtree {                                                        451a
  int l[MAXN], m[MAXN], r[MAXN];                                        27be
  LL val[MAXN], tadd[MAXN], tmul[MAXN];                                 4510
                                                                        427e
#define lson (o<<1)                                                     ac35
#define rson (o<<1|1)                                                   1294
                                                                        427e
  void pull(int o) {                                                    1344
    val[o] = (val[lson] + val[rson]) % p;                               bbe9
  }                                                                     95cf
                                                                        427e
  void push_add(int o, LL x) {                                          e4bc
    val[o] = (val[o] + x * (r[o] - l[o])) % p;                          5dd6
    tadd[o] = (tadd[o] + x) % p;                                        6eff
  }                                                                     95cf
                                                                        427e
  void push_mul(int o, LL x) {                                          d658
    val[o] = val[o] * x % p;                                            b82c
    tadd[o] = tadd[o] * x % p;                                          aa86
    tmul[o] = tmul[o] * x % p;                                          649f
  }                                                                     95cf
                                                                        427e
  void push(int o) {                                                    b149
    if (l[o] == m[o]) return;                                           3159
    if (tmul[o] != 1) {                                                 0a90
      push_mul(lson, tmul[o]);                                          0f4a
      push_mul(rson, tmul[o]);                                          045e
      tmul[o] = 1;                                                      ac0a
    }                                                                   95cf
    if (tadd[o]) {                                                      1b82
      push_add(lson, tadd[o]);                                          9547
      push_add(rson, tadd[o]);                                          0e73
      tadd[o] = 0;                                                      6234
    }                                                                   95cf
  }                                                                     95cf
                                                                        427e
  void build(int o, int ll, int rr) {                                   471c
    int mm = (ll + rr) / 2;                                             0e87
    l[o] = ll; r[o] = rr; m[o] = mm;                                    9d27
```

```
    tmul[o] = 1;
    if (ll == mm) {
      scanf("%lld", val + o);
      val[o] %= p;
    } else {
      build(lson, ll, mm);
      build(rson, mm, rr);
      pull(o);
    }
  }

  void add(int o, int ll, int rr, LL x) {
    if (ll <= l[o] && r[o] <= rr) {
      push_add(o, x);
    } else {
      push(o);
      if (m[o] > ll) add(lson, ll, rr, x);
      if (m[o] < rr) add(rson, ll, rr, x);
      pull(o);
    }
  }

  void mul(int o, int ll, int rr, LL x) {
    if (ll <= l[o] && r[o] <= rr) {
      push_mul(o, x);
    } else {
      push(o);
      if (ll < m[o]) mul(lson, ll, rr, x);
      if (m[o] < rr) mul(rson, ll, rr, x);
      pull(o);
    }
  }

  LL query(int o, int ll, int rr) {
    if (ll <= l[o] && r[o] <= rr) {
      return val[o];
    } else {
      push(o);
      if (rr <= m[o]) return query(lson, ll, rr);
      if (ll >= m[o]) return query(rson, ll, rr);
      return query(lson, ll, rr) + query(rson, ll, rr);
    }
  }
} seg;
```

6.4 Link/cut tree

Usage:

`pull(x)`	Collect information of subtrees.
`Link(u, v)`	Link two unconnected trees.
`Cut(u, v)`	Cut an existent edge.
`Query(u, v)`	Path aggregation.
`Update(u, x)`	Single point modification.

```
// about 0.13s per 100k ops @Luogu.org

namespace LCT {
  const int MAXN = 300005;
  int fa[MAXN], ch[MAXN][2], val[MAXN], sum[MAXN];
  bool rev[MAXN];

  bool isroot(int x) {
    return ch[fa[x]][0] == x || ch[fa[x]][1] == x;
  }

  void pull(int x) {
    sum[x] = val[x] ^ sum[ch[x][0]] ^ sum[ch[x][1]];
  }

  void reverse(int x) {
    swap(ch[x][0], ch[x][1]);
    rev[x] ^= 1;
  }

  void push(int x) {
    if (rev[x]) {
      if (ch[x][0]) reverse(ch[x][0]);
      if (ch[x][1]) reverse(ch[x][1]);
      rev[x] = 0;
    }
  }

  void rotate(int x) {
    int y = fa[x], z = fa[y], k = ch[y][1] == x, w = ch[x][!k];
    if (isroot(y)) ch[z][ch[z][1] == y] = x;
    ch[x][!k] = y; ch[y][k] = w;
```

```
    if (w) fa[w] = y;
    fa[y] = x; fa[x] = z;
    pull(y);
  }

  void pushall(int x) {
    if (isroot(x)) pushall(fa[x]);
    push(x);
  }

  void splay(int x) {
    int y = x, z = 0;
    pushall(y);
    while (isroot(x)) {
      y = fa[x]; z = fa[y];
      if (isroot(y)) rotate((ch[y][0] == x) ^ (ch[z][0] == y) ? x : y);
      rotate(x);
    }
    pull(x);
  }

  void access(int x) {
    int z = x;
    for (int y = 0; x; x = fa[y = x]) {
      splay(x);
      ch[x][1] = y;
      pull(x);
    }
    splay(z);
  }

  void chroot(int x) {
    access(x);
    reverse(x);
  }

  void split(int x, int y) {
    chroot(x);
    access(y);
  }

  int Root(int x) {
    access(x);
    while (ch[x][0]) {
      push(x);
      x = ch[x][0];
    }
    splay(x);
    return x;
  }

  void Link(int u, int v) {  // assume unconnected before
    chroot(u);
    fa[u] = v;
  }

  void Cut(int u, int v) {  // assume connected before
    split(u, v);
    fa[u] = ch[v][0] = 0;
    pull(v);
  }

  int Query(int u, int v) {
    split(u, v);
    return sum[v];
  }

  void Update(int u, int x) {
    splay(u);
    val[u] = x;
  }
};
```

6.5 Balanced binary search tree from pb_ds

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;

tree<int, null_type, less<int>, rb_tree_tag, tree_order_statistics_node_update>
  rkt;
// null_tree_node_update

// SAMPLE USAGE
rkt.insert(x);          // insert element
rkt.erase(x);           // erase element
rkt.order_of_key(x);    // obtain the number of elements less than x
```

```
b064 rkt.find_by_order(i);   // iterator to i-th (numbered from 0) smallest element
c103 rkt.lower_bound(x);
4ff4 rkt.upper_bound(x);
b19b rkt.join(rkt2);         // merge tree (only if their ranges do not intersect)
cb47 rkt.split(x, rkt2);     // split all elements greater than x to rkt2
```

6.6 Persistent segment tree, range k-th query

```
f1a7 struct node {
2ff6   static int n, pos;
427e
7cec   int value;
70e2   node *left, *right;
427e
20b0   void* operator new(size_t size);
427e
3dc0   static node* Build(int l, int r) {
b6c5     node* a = new node;
ce96     if (r > l + 1) {
181e       int mid = (l + r) / 2;
3ba2       a->left = Build(l, mid);
8aaf       a->right = Build(mid, r);
8e2e     } else {
bfc4       a->value = 0;
95cf     }
5ffd     return a;
95cf   }
427e
5a45   static node* init(int size) {
2c46     n = size;
7ee3     pos = 0;
be52     return Build(0, n);
95cf   }
427e
93c0   static int Query(node* lt, node *rt, int l, int r, int k) {
d30c     if (r == l + 1) return l;
181e     int mid = (l + r) / 2;
cb5a     if (rt->left->value - lt->left->value < k) {
8edb       k -= rt->left->value - lt->left->value;
2412       return Query(lt->right, rt->right, mid, r, k);
8e2e     } else {
0119       return Query(lt->left, rt->left, l, mid, k);
95cf     }
95cf   }
427e
c9ad   static int query(node* lt, node *rt, int k) {
9e27     return Query(lt, rt, 0, n, k);
95cf   }
427e
b19c   node *Inc(int l, int r, int pos) const {
5794     node* a = new node(*this);
ce96     if (r > l + 1) {
181e       int mid = (l + r) / 2;
203d       if (pos < mid)
f44a         a->left = left->Inc(l, mid, pos);
649a       else
1024         a->right = right->Inc(mid, r, pos);
95cf     }
2b3e     a->value++;
5ffd     return a;
95cf   }
427e
e80f   node *inc(int index) {
c246     return Inc(0, n, index);
95cf   }
865a } nodes[8000000];
427e
99ce int node::n, node::pos;
1987 inline void* node::operator new(size_t size) {
bb3c   return nodes + (pos++);
95cf }
```

6.7 Sparse table, range extremum query

The array is 0-based and the range is closed.

```
db63 const int MAXN = 100007;
b330 int a[MAXN];
69ae int st[MAXN][32 - __builtin_clz(MAXN)];
427e
8041 inline int ext(int x, int y){return x>y?x:y;} // ! max
427e
d34f void init(int n){
ce01     int l = 31 - __builtin_clz(n);
cf75     rep (i, n) st[i][0] = a[i];
```

```
    rep (j, l)
        rep (i, 1+n-(1<<j))
            st[i][j+1] = ext(st[i][j], st[i+(1<<j)][j]);
}

int rmq(int l, int r){
    int k = 31 - __builtin_clz(r-l+1);
    return ext(st[l][k], st[r-(1<<k)+1][k]);
}
```

7 Geometrics

7.1 2D geometric template

```
#include <bits/stdc++.h>
using namespace std;

typedef int T;
typedef struct pt {
    T x, y;
    T operator , (pt a) { return x*a.x + y*a.y; } // inner product
    T operator * (pt a) { return x*a.y - y*a.x; } // outer product
    pt operator + (pt a) { return {x+a.x, y+a.y}; }
    pt operator - (pt a) { return {x-a.x, y-a.y}; }

    pt operator * (T k) { return {x*k, y*k}; }
    pt operator - () { return {-x, -y};}
} vec;

typedef pair<pt, pt> seg;

bool ptOnSeg(pt& p, seg& s){
    vec v1 = s.first - p, v2 = s.second - p;
    return (v1, v2) <= 0 && v1 * v2 == 0;
}

// 0 not on segment
// 1 on segment except vertices
// 2 on vertices
int ptOnSeg2(pt& p, seg& s){
    vec v1 = s.first - p, v2 = s.second - p;
    T ip = (v1, v2);
    if (v1 * v2 != 0 || ip > 0) return 0;
    return (v1, v2) ? 1 : 2;
}

// if two orthogonal rectangles do not touch, return true
inline bool nIntRectRect(seg a, seg b){
    return min(a.first.x, a.second.x) > max(b.first.x, b.second.x) ||
           min(a.first.y, a.second.y) > max(b.first.y, b.second.y) ||
           min(b.first.x, b.second.x) > max(a.first.x, a.second.x) ||
           min(b.first.y, b.second.y) > max(a.first.y, a.second.y);
}

// >0 in order
// <0 out of order
// =0 not standard
inline double rotOrder(vec a, vec b, vec c){return double(a*b)*(b*c);}

inline bool intersect(seg a, seg b){
    // ! if (nIntRectRect(a, b)) return false; // if commented, assume that a
      and b are non-collinear
    return rotOrder(b.first-a.first, a.second-a.first, b.second-a.first) >= 0 &&
           rotOrder(a.first-b.first, b.second-b.first, a.second-b.first) >= 0;
}

// 0 not insersect
// 1 standard intersection
// 2 vertex-line intersection
// 3 vertex-vertex intersection
// 4 collinear and have common point(s)
int intersect2(seg& a, seg& b){
    if (nIntRectRect(a, b)) return 0;
    vec va = a.second - a.first, vb = b.second - b.first;
    double j1 = rotOrder(b.first-a.first, va, b.second-a.first),
           j2 = rotOrder(a.first-b.first, vb, a.second-b.first);
    if (j1 < 0 || j2 < 0) return 0;
    if (j1 != 0 && j2 != 0) return 1;
    if (j1 == 0 && j2 == 0){
        if (va * vb == 0) return 4; else return 3;
    } else return 2;
}

template <typename Tp = T>
inline pt getIntersection(pt P, vec v, pt Q, vec w){
```

```
6850     static_assert(is_same<Tp, double>::value, "must be double!");
7c9a     return P + v * (w*(P-Q)/(v*w));
95cf }
427e
427e // -1 outside the polygon
427e // 0  on the border of the polygon
427e // 1  inside the polygon
cbdd int ptOnPoly(pt p, pt* poly, int n){
5fb4     int wn = 0;
1294     for (int i = 0; i < n; i++) {
427e
3cae         T k, d1 = poly[i].y - p.y, d2 = poly[(i+1)%n].y - p.y;
b957         if (k = (poly[(i+1)%n] - poly[i])*(p - poly[i])){
8c40             if (k > 0 && d1 <= 0 && d2 > 0) wn++;
3c4d             if (k < 0 && d2 <= 0 && d1 > 0) wn--;
aad3         } else return 0;
95cf     }
0a5f     return wn ? 1 : -1;
95cf }
427e
d4a3 istream& operator >> (istream& lhs, pt& rhs){
fa86     lhs >> rhs.x >> rhs.y;
331a     return lhs;
95cf }
427e
07ae istream& operator >> (istream& lhs, seg& rhs){
5cab     lhs >> rhs.first >> rhs.second;
331a     return lhs;
95cf }
```

8 Appendices

8.1 Primes

8.1.1 First primes

p	$g(p)$	p	$g(p)$	p	$g(p)$	p	$g(p)$	p	$g(p)$
2	1	3	2	5	2	7	3	11	2
13	2	17	3	19	2	23	5	29	2
31	3	37	2	41	6	43	3	47	5
53	2	59	2	61	2	67	2	71	7
73	5	79	3	83	2	89	3	97	5
101	2	103	5	107	2	109	6	113	3
127	3	131	2	137	3	139	2	149	2
151	6	157	5	163	2	167	5	173	2
179	2	181	2	191	19	193	5	197	2
199	3	211	2	223	3	227	2	229	6

8.1.2 Arbitrary length primes

$\lg p$	p	$g(p)$	p	$g(p)$
3	967	5	1031	14
4	9859	2	10273	10
5	96331	10	102931	3
6	958543	6	1031137	5
7	9594539	2	10169651	2
8	96243449	3	103211039	7
9	980483981	2	1042484357	2
10	9858935453	2	10261276009	7
11	95748666809	3	101759940101	2
12	950781833849	3	1012797784423	5
13	9739822952371	7	10037217092377	7
14	96181051140397	5	104974966380359	11
15	981030138360889	13	1029038416465403	2
16	9655206098080843	3	10116299875820773	2
17	97687777921994419	3	101506415998163437	2

8.1.3 $\sim 1 \times 10^9$

p	$g(p)$	p	$g(p)$	p	$g(p)$
954854573	3	967607731	2	973215833	3
975831713	3	978949117	2	980766497	3
983879921	3	985918807	3	986608921	29
991136977	5	991752599	13	997137961	11
1003911991	3	1009775293	2	1012423549	6
1021000537	5	1023976897	7	1024153643	2
1037027287	3	1038812881	11	1044754639	3
1045125617	3	1047411427	3	1047753349	6

8.1.4 $\sim 1 \times 10^{18}$

p	$g(p)$	p	$g(p)$
951970612352230049	3	963284339889659609	3
967495386904694119	3	969751761517096213	2
983238274281901499	2	984647442475101409	23
989286107138674069	11	1002507954383424641	3
1006658951440146419	2	1020152326159075903	3
1034876265966119449	7	1042753851435034019	2
1043609016597371563	2	1045571042176595707	2
1048364250160580293	2	1049495624119026949	2

8.2 Pell's equation

$x^2 - ny^2 = 1$, where n is a positive nonsquare integer.
Let (x_0, y_0) be the smallest positive solution of the equation, then the k-th solution is:

$$\begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} x_0 & ny_0 \\ y_0 & x_0 \end{pmatrix}^k \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

Some smallest solutions to Pell's equation:

n	2	3	5	6	7	8	10	11	12	13	14	15	17	18	19	20
x	3	2	9	5	8	3	19	10	7	649	15	4	33	17	170	9
y	2	1	4	2	3	1	6	3	2	180	4	1	8	4	39	2

8.3 Burnside's lemma and Polya's enumeration theorem

The Burnside's lemma says that

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$$

where G is a group acting on X, X^g is the set of elements in X that are fixed by g, i.e. $X^g = \{x \in X : gx = x\}$.
The unweighted version of Pólya enumeration theorem says that

$$|Y^X/G| = \frac{1}{|G|} \sum_{g \in G} m^{c_g}$$

where $m = |X|$ is the number of colors, c_g is the number of the cycles of permutation g.

8.4 Lagrange's interpolation

For sample points $(x_0, y_0), \cdots, (x_k, y_k)$, define

$$l_j(x) = \prod_{0 \le m \le k, m \ne j} fracx - x_m x_j - x_m$$

then the Lagrange polynomial is

$$L(x) = \sum_{j=0}^{k} y_j l_j(x).$$