Team notebook

HCMUS-PenguinSpammers

January 28, 2022

Contents

1 Algorithms **1**
- 1.1 Mo's Algorithm 1
- 1.2 Mo's Algorithms on Trees 1
- 1.3 Parallel Binary Search 2

2 Combinatorics **2**
- 2.1 Factorial Approximate 2
- 2.2 Factorial 2
- 2.3 Fast Fourier Transform 2
- 2.4 General purpose numbers 3
- 2.5 Lucas Theorem 4
- 2.6 Multinomial 4
- 2.7 Others 4
- 2.8 Permutation To Int 5
- 2.9 Sigma Function 5

3 Data Structures **5**
- 3.1 Binary Index Tree 5
- 3.2 Disjoint Set Uninon (DSU) 5
- 3.3 Fake Update 5
- 3.4 Fenwick Tree 6
- 3.5 Hash Table 6
- 3.6 Range Minimum Query 7
- 3.7 STL Treap 7
- 3.8 Segment Tree 7
- 3.9 Sparse Table 8
- 3.10 Trie 8

4 Dynamic Programming Optimization **8**
- 4.1 Convex Hull Trick 8
- 4.2 Divide and Conquer 9

1 Algorithms

1.1 Mo's Algorithm

```
/*
   https://www.spoj.com/problems/FREQ2/
*/
vector <int> MoQueries(int n, vector <query> Q){

   block_size = sqrt(n);
   sort(Q.begin(), Q.end(), [](const query &A,
       const query &B){
      return (A.l/block_size != B.l/block_size)?
          (A.l/block_size < B.l/block_size) :
          (A.r < B.r);
   });
   vector <int> res;
   res.resize((int)Q.size());

   int L = 1, R = 0;
   for(query q: Q){
      while (L > q.l) add(--L);
      while (R < q.r) add(++R);

      while (L < q.l) del(L++);
      while (R > q.r) del(R--);

      res[q.pos] = calc(1, R-L+1);
   }
   return res;
}
```

1.2 Mo's Algorithms on Trees

```
/*
Given a tree with N nodes and Q queries. Each
    node has an integer weight.
Each query provides two numbers u and v, ask for
    how many different integers weight of nodes
there are on path from u to v.

----------
Modify DFS:
----------
For each node u, maintain the start and the end
    DFS time. Let's call them ST(u) and EN(u).
=> For each query, a node is considered if its
    occurrence count is one.

--------------
Query solving:
--------------
Let's query be (u, v). Assume that ST(u) <=
    ST(v). Denotes P as LCA(u, v).

Case 1: P = u
```

```
Our query would be in range [ST(u), ST(v)].

Case 2: P != u
Our query would be in range [EN(u), ST(v)] +
    [ST(p), ST(p)]
*/

void update(int &L, int &R, int qL, int qR){
    while (L > qL) add(--L);
    while (R < qR) add(++R);

    while (L < qL) del(L++);
    while (R > qR) del(R--);
}

vector <int> MoQueries(int n, vector <query> Q){
    block_size = sqrt((int)nodes.size());
    sort(Q.begin(), Q.end(), [](const query &A,
        const query &B){
        return (ST[A.l]/block_size !=
            ST[B.l]/block_size)?
            (ST[A.l]/block_size <
            ST[B.l]/block_size) : (ST[A.r] <
            ST[B.r]);
    });
    vector <int> res;
    res.resize((int)Q.size());

    LCA lca;
    lca.initialize(n);

    int L = 1, R = 0;
    for(query q: Q){
        int u = q.l, v = q.r;
        if(ST[u] > ST[v]) swap(u, v); // assume
            that S[u] <= S[v]
        int parent = lca.get(u, v);

        if(parent == u){
            int qL = ST[u], qR = ST[v];
            update(L, R, qL, qR);
        }else{
            int qL = EN[u], qR = ST[v];
            update(L, R, qL, qR);
            if(cnt_val[a[parent]] == 0)
                res[q.pos] += 1;
        }

        res[q.pos] += cur_ans;
    }
    return res;
}
```

1.3 Parallel Binary Search

```
int lo[N], mid[N], hi[N];
vector<int> vec[N];

void clear() //Reset
{
        memset(bit, 0, sizeof(bit));
}

void apply(int idx) //Apply ith update/query
{
        if(ql[idx] <= qr[idx])
                update(ql[idx], qa[idx]),
                    update(qr[idx]+1, -qa[idx]);
        else
        {
                update(1, qa[idx]);
                update(qr[idx]+1, -qa[idx]);
                update(ql[idx], qa[idx]);
        }
}

bool check(int idx) //Check if the condition is
    satisfied
{
        int req=reqd[idx];
        for(auto &it:owns[idx])
        {
                req-=pref(it);
                if(req<0)
                        break;
        }
        if(req<=0)
                return 1;
        return 0;
}

void work()
{
        for(int i=1;i<=q;i++)
                vec[i].clear();
        for(int i=1;i<=n;i++)
                if(mid[i]>0)
                        vec[mid[i]].push_back(i);
        clear();
        for(int i=1;i<=q;i++)
        {
                apply(i);
                for(auto &it:vec[i]) //Add
                    appropriate check conditions
                {
                        if(check(it))
                                hi[it]=i;
                        else
                                lo[it]=i+1;
                }
        }
}

void parallel_binary()
{
        for(int i=1;i<=n;i++)
                lo[i]=1, hi[i]=q+1;
        bool changed = 1;
        while(changed)
        {
                changed=0;
                for(int i=1;i<=n;i++)
                {
                        if(lo[i]<hi[i])
                        {
                                changed=1;
                                mid[i]=(lo[i] +
                                    hi[i])/2;
                        }
                        else
```

```
                    mid[i]=-1;
            }
            work();
        }
}
```

2 Combinatorics

2.1 Factorial Approximate

Approximate Factorial:

$$n! = \sqrt{2.\pi.n}.(\frac{n}{e})^n \tag{1}$$

2.2 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
n	11	12	13	14	15	16	17			
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14			
n	20	25	30	40	50	100	150	171		
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX		

2.3 Fast Fourier Transform

```
/**
 * Fast Fourier Transform.
 * Useful to compute convolutions.
 * computes:
 *    C(f star g)[n] = sum_m(f[m] * g[n - m])
 * for all n.
 * test: icpc live archive, 6886 - Golf Bot
 * */


using namespace std;
#include <bits/stdc++.h>
#define D(x) cout << #x " = " << (x) << endl
#define endl '\n'

const int MN = 262144 << 1;
int d[MN + 10], d2[MN + 10];


const double PI = acos(-1.0);

struct cpx {
  double real, image;
  cpx(double _real, double _image) {
    real = _real;
    image = _image;
  }
  cpx(){}
};

cpx operator + (const cpx &c1, const cpx &c2) {
  return cpx(c1.real + c2.real, c1.image +
      c2.image);
}

cpx operator - (const cpx &c1, const cpx &c2) {
  return cpx(c1.real - c2.real, c1.image -
      c2.image);
}

cpx operator * (const cpx &c1, const cpx &c2) {
  return cpx(c1.real*c2.real - c1.image*c2.image,
      c1.real*c2.image + c1.image*c2.real);
}

int rev(int id, int len) {
  int ret = 0;
  for (int i = 0; (1 << i) < len; i++) {
    ret <<= 1;
    if (id & (1 << i)) ret |= 1;
  }
  return ret;
}

cpx A[1 << 20];

void FFT(cpx *a, int len, int DFT) {
  for (int i = 0; i < len; i++)
    A[rev(i, len)] = a[i];
  for (int s = 1; (1 << s) <= len; s++) {
    int m = (1 << s);
    cpx wm = cpx(cos( DFT * 2 * PI / m), sin(DFT
        * 2 * PI / m));
    for(int k = 0; k < len; k += m) {
      cpx w = cpx(1, 0);
      for(int j = 0; j < (m >> 1); j++) {
        cpx t = w * A[k + j + (m >> 1)];
        cpx u = A[k + j];
        A[k + j] = u + t;
        A[k + j + (m >> 1)] = u - t;
        w = w * wm;
      }
    }
  }
  if (DFT == -1) for (int i = 0; i < len; i++)
      A[i].real /= len, A[i].image /= len;
  for (int i = 0; i < len; i++) a[i] = A[i];
  return;
}

cpx in[1 << 20];

void solve(int n) {
  memset(d, 0, sizeof d);
  int t;
  for (int i = 0; i < n; ++i) {
    cin >> t;
    d[t] = true;
  }
  int m;
  cin  >> m;
  vector<int> q(m);
  for (int i = 0; i < m; ++i)
    cin >> q[i];

  for (int i = 0; i < MN; ++i) {
    if (d[i])
      in[i] = cpx(1, 0);
    else
      in[i] = cpx(0, 0);
  }
```

```
  FFT(in, MN, 1);
  for (int i = 0; i < MN; ++i) {
    in[i] = in[i] * in[i];
  }
  FFT(in, MN, -1);

  int ans = 0;
  for (int i = 0; i < q.size(); ++i) {
    if (in[q[i]].real > 0.5 || d[q[i]]) {
      ans++;
    }
  }
  cout << ans << endl;
}

int main() {
  ios_base::sync_with_stdio(false);cin.tie(NULL);
  int n;
  while (cin >> n)
    solve(n);
  return 0;
}
```

2.4 General purpose numbers

Bernoulli numbers
EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t-1}$ (FFT-able). $B[0,\dots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \dots]$

Sums of powers:

$$\sum_{i=1}^{n} n^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k\cdot(n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^{\infty} f(x)dx - \sum_{k=1}^{\infty}\frac{B_k}{k!}f^{(k-1)}(m)$$

$$\approx \int_m^{\infty} f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k)$, $c(0,0) = 1$
$\sum_{k=0}^{n} c(n,k)x^k = x(x+1)\dots(x+n-1)$

$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$

Stirling numbers of the second kind
Partitions of n distinct elements into exactly k groups.

$$S(n,k) = S(n-1,k-1) + kS(n-1,k)$$

$$S(n,1) = S(n,n) = 1$$

$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

Eulerian numbers
Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$

$$E(n,0) = E(n,n-1) = 1$$

$$E(n,k) = \sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$$

Bell numbers
Total number of partitions of n distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \dots$. For p prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod{p}$$

Labeled unrooted trees
\# on n vertices: n^{n-2}
\# on k existing trees of size n_i: $n_1 n_2 \cdots n_k n^{k-2}$
\# with degrees d_i: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2}C_n,\ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \dots$

[noitemsep]sub-diagonal monotone paths in an $n \times n$ grid. strings with n pairs of parenthesis, correctly nested. binary trees with with $n+1$ leaves (0 or 2 children). ordered trees with $n+1$ vertices. ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines. permutations of $[n]$ with no 3-term increasing subseq.

2.5 Lucas Theorem

For non-negative integers m and n and a prime p, the following congruence relation holds: :

$$\binom{m}{n} \equiv \prod_{i=0}^{k}\binom{m_i}{n_i} \pmod{p},$$

where :

$$m = m_k p^k + m_{k-1}p^{k-1} + \cdots + m_1 p + m_0,$$

and :

$$n = n_k p^k + n_{k-1}p^{k-1} + \cdots + n_1 p + n_0$$

are the base p expansions of m and n respectively. This uses the convention that $\binom{m}{n} = 0$ if $m \leq n$.

2.6 Multinomial

```
/**
 * Description: Computes $\displaystyle
     \binom{k_1 + \dots + k_n}{k_1, k_2, \dots,
     k_n} = \frac{(\sum k_i)!}{k_1!k_2!...k_n!}$.
 * Status: Tested on kattis:lexicography
 */
#pragma once
```

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

2.7 Others

Cycles Let $g_S(n)$ be the number of n-permutations whose cycle lengths all belong to the set S. Then

$$\sum_{n=0}^{\infty} g_S(n)\frac{x^n}{n!} = \exp\left(\sum_{n\in S}\frac{x^n}{n}\right)$$

Derangements Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1)+D(n-2)) = nD(n-1)+(-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$$

Burnside's lemma Given a group G of symmetries and a set X, the number of elements of X *up to symmetry* equals

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

where X^g are the elements fixed by g ($g.x = x$).

If $f(n)$ counts "configurations" (of some sort) of length n, we can ignore rotational symmetry using $G = Z_n$ to get

$$g(n) = \frac{1}{n}\sum_{k=0}^{n-1} f(\gcd(n,k)) = \frac{1}{n}\sum_{k|n} f(k)\phi(n/k).$$

2.8 Permutation To Int

```
/**
 * Description: Permutation -> integer
     conversion. (Not order preserving.)
 * Integer -> permutation can use a lookup table.
 * Time: O(n)
 **/

int permToInt(vector<int>& v) {
	int use = 0, i = 0, r = 0;
	for(int x : v) r = r * ++i +
	     __builtin_popcount(use & -(1<<x)),
	        use |= 1 << x;                  //
	              (note: minus, not ~!)
	return r;
}
```

2.9 Sigma Function

The Sigma Function is defined as:

$$\sigma_x(n) = \sum_{d|n} d^x$$

when $x = 0$ is called the divisor function, that counts the number of positive divisors of n.

Now, we are interested in find

$$\sum_{d|n} \sigma_0(d)$$

If n is written as prime factorization:

$$n = \prod_{i=1}^{k} P_i^{e_k}$$

We can demonstrate that:

$$\sum_{d|n} \sigma_0(d) = \prod_{i=1}^{k} g(e_k + 1)$$

where $g(x)$ is the sum of the first x positive numbers:

$$g(x) = (x * (x + 1))/2$$

3 Data Structures

3.1 Binary Index Tree

```
struct BIT {
     int n;
     int t[2 * N];

     void add(int where, long long what) {
        for (where++; where <= n; where += where &
             -where) {
            t[where] += what;
        }
     }

     void add(int from, int to, long long what) {
         add(from, what);
         add(to + 1, -what);
     }

     long long query(int where) {
         long long sum = t[0];

         for (where++; where > 0; where -= where &
              -where) {
             sum += t[where];
         }

         return sum;
     }
};
```

3.2 Disjoint Set Uninon (DSU)

```
class DSU{
public:
    vector <int> parent;
    void initialize(int n){
        parent.resize(n+1, -1);
    }

    int findSet(int u){
        while(parent[u] > 0)
            u = parent[u];
        return u;
    }

    void Union(int u, int v){
        int x = parent[u] + parent[v];
        if(parent[u] > parent[v]){
            parent[v] = x;
            parent[u] = v;
        }else{
            parent[u] = x;
            parent[v] = u;
        }
    }
};
```

3.3 Fake Update

```
vector <int> fake_bit[MAXN];

void fake_update(int x, int y, int limit_x){
    for(int i = x; i < limit_x; i += i&(-i))
        fake_bit[i].pb(y);
}

void fake_get(int x, int y){
    for(int i = x; i >= 1; i -= i&(-i))
        fake_bit[i].pb(y);
}
vector <int> bit[MAXN];

void update(int x, int y, int limit_x, int val){
    for(int i = x; i < limit_x; i += i&(-i)){
        for(int j =
             lower_bound(fake_bit[i].begin(),
             fake_bit[i].end(), y) -
             fake_bit[i].begin(); j <
             fake_bit[i].size(); j += j&(-j))
            bit[i][j] = max(bit[i][j], val);
    }
}

int get(int x, int y){
    int ans = 0;
    for(int i = x; i >= 1; i -= i&(-i)){
        for(int j =
             lower_bound(fake_bit[i].begin(),
             fake_bit[i].end(), y) -
             fake_bit[i].begin(); j >= 1; j -=
             j&(-j))
            ans = max(ans, bit[i][j]);
    }
    return ans;
}

int main(){
    _io
    int n; cin >> n;
    vector <int> Sx, Sy;
    for(int i = 1; i <= n; i++){
        cin >> a[i].fi >> a[i].se;
        Sx.pb(a[i].fi);
        Sy.pb(a[i].se);
    }
    unique_arr(Sx);
    unique_arr(Sy);
    // unique all value
    for(int i = 1; i <= n; i++){
        a[i].fi = lower_bound(Sx.begin(),
             Sx.end(), a[i].fi) - Sx.begin();
        a[i].se = lower_bound(Sy.begin(),
             Sy.end(), a[i].se) - Sy.begin();
    }
    // do fake BIT update and get operator
    for(int i = 1; i <= n; i++){
        fake_get(a[i].fi-1, a[i].se-1);
        fake_update(a[i].fi, a[i].se,
            (int)Sx.size());
    }

    for(int i = 0; i < Sx.size(); i++){
        fake_bit[i].pb(INT_MIN); // avoid zero
        sort(fake_bit[i].begin(),
            fake_bit[i].end());
        fake_bit[i].resize(unique(fake_bit[i].begin(),
            fake_bit[i].end()) -
            fake_bit[i].begin());
        bit[i].resize((int)fake_bit[i].size(), 0);
    }

    // real update, get operator
    int res = 0;
    for(int i = 1; i <= n; i++){
        int maxCurLen = get(a[i].fi-1, a[i].se-1)
            + 1;
        res = max(res, maxCurLen);
        update(a[i].fi, a[i].se, (int)Sx.size(),
            maxCurLen);
    }
}
```

3.4 Fenwick Tree

```
template <typename T>
class FenwickTree{
  vector <T> fenw;
  int n;
public:
  void initialize(int _n){
    this->n = _n;
    fenw.resize(n+1);
  }

  void update(int id, T val) {
```

```
    while (id <= n) {
      fenw[id] += val;
      id += id&(-id);
      }
  }

  T get(int id){
    T ans{};
    while(id >= 1){
      ans += fenw[id];
      id -= id&(-id);
    }
    return ans;
  }
};
```

3.5 Hash Table

```
/*
 * Micro hash table, can be used as a set.
 *  Very efficient vs std::set
 *
 */

const int MN = 1001;
struct ht {
  int _s[(MN + 10) >> 5];
  int len;
  void set(int id) {
    len++;
    _s[id >> 5] |= (1LL << (id & 31));
  }
  bool is_set(int id) {
    return _s[id >> 5] & (1LL << (id & 31));
  }
};
```

3.6 Range Minimum Query

```
/*
    return min(v[a], v[a + 1], ..., v[b - 1]) in
        constant time
*/

template<class T>
struct RMQ {
        vector<vector<T>> jmp;
        RMQ(const vector<T>& V) : jmp(1, V) {
                for (int pw = 1, k = 1; pw * 2 <=
                    sz(V); pw *= 2, ++k) {
                        jmp.emplace_back(sz(V) - pw
                            * 2 + 1);
                        rep(j,0,sz(jmp[k]))
                                jmp[k][j] =
                                    min(jmp[k -
                                    1][j], jmp[k -
                                    1][j + pw]);
                }
        }
        T query(int a, int b) {
                assert(a < b); // or return inf if
                    a == b
                int dep = 31 - __builtin_clz(b - a);
                return min(jmp[dep][a], jmp[dep][b
                    - (1 << dep)]);
        }
};
```

3.7 STL Treap

```
struct Node {
        Node *l = 0, *r = 0;
        int val, y, c = 1;
        Node(int val) : val(val), y(rand()) {}
        void recalc();
};

int cnt(Node* n) { return n ? n->c : 0; }
void Node::recalc() { c = cnt(l) + cnt(r) + 1; }

template<class F> void each(Node* n, F f) {
        if (n) { each(n->l, f); f(n->val);
            each(n->r, f); }
}

pair<Node*, Node*> split(Node* n, int k) {
        if (!n) return {};
        if (cnt(n->l) >= k) { // "n->val >= k" for
            lower_bound(k)
                auto pa = split(n->l, k);
                n->l = pa.second;
                n->recalc();
                return {pa.first, n};
        } else {
                auto pa = split(n->r, k - cnt(n->l)
                    - 1); // and just "k"
                n->r = pa.first;
                n->recalc();
                return {n, pa.second};
        }
}

Node* merge(Node* l, Node* r) {
        if (!l) return r;
        if (!r) return l;
        if (l->y > r->y) {
                l->r = merge(l->r, r);
                l->recalc();
                return l;
        } else {
                r->l = merge(l, r->l);
                r->recalc();
                return r;
        }
}

Node* ins(Node* t, Node* n, int pos) {
        auto pa = split(t, pos);
        return merge(merge(pa.first, n),
            pa.second);
}

// Example application: move the range [l, r) to
    index k
void move(Node*& t, int l, int r, int k) {
```

```
        Node *a, *b, *c;
        tie(a,b) = split(t, l); tie(b,c) =
            split(b, r - l);
        if (k <= l) t = merge(ins(a, b, k), c);
        else t = merge(a, ins(c, b, k - r));
}
```

3.8 Segment Tree

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

const int N = 1e5 + 10;

int node[4*N];

void modify(int seg, int l, int r, int p, int
     val){
    if(l == r){
        node[seg] += val;
        return;
    }
    int mid = (l + r)/2;
    if(p <= mid){
        modify(2*seg + 1, l, mid, p, val);
    }else{
        modify(2*seg + 2, mid + 1, r, p, val);
    }
    node[seg] = node[2*seg + 1] + node[2*seg + 2];
}

int sum(int seg, int l, int r, int a, int b){
    if(l > b || r < a) return 0;
    if(l >= a && r <= b) return node[seg];
    int mid = (l + r)/2;
    return sum(2*seg + 1, l, mid, a, b) +
         sum(2*seg + 2, mid + 1, r, a, b);
}
```

3.9 Sparse Table

```
template <typename T, typename func =
     function<T(const T, const T)>>
struct SparseTable {
    func calc;
    int n;
    vector<vector<T>> ans;

    SparseTable() {}

    SparseTable(const vector<T>& a, const func&
         f) : n(a.size()), calc(f) {
        int last = trunc(log2(n)) + 1;
        ans.resize(n);
        for (int i = 0; i < n; i++){
            ans[i].resize(last);
        }
        for (int i = 0; i < n; i++){
            ans[i][0] = a[i];
        }
        for (int j = 1; j < last; j++){
            for (int i = 0; i <= n - (1 << j);
                 i++){
                ans[i][j] = calc(ans[i][j - 1],
                     ans[i + (1 << (j - 1))][j -
                     1]);
            }
        }
    }

    T query(int l, int r){
        assert(0 <= l && l <= r && r < n);
        int k = trunc(log2(r - l + 1));
        return calc(ans[l][k], ans[r - (1 << k) +
             1][k]);
    }
};
```

3.10 Trie

```
const int MN = 26; // size of alphabet
const int MS = 100010; // Number of states.

struct trie{
  struct node{
    int c;
    int a[MN];
  };

  node tree[MS];
  int nodes;

  void clear(){
    tree[nodes].c = 0;
    memset(tree[nodes].a, -1, sizeof
         tree[nodes].a);
    nodes++;
  }

  void init(){
    nodes = 0;
    clear();
  }

  int add(const string &s, bool query = 0){
    int cur_node = 0;
    for(int i = 0; i < s.size(); ++i){
      int id = gid(s[i]);
      if(tree[cur_node].a[id] == -1){
        if(query) return 0;
        tree[cur_node].a[id] = nodes;
        clear();
      }
      cur_node = tree[cur_node].a[id];
    }
    if(!query) tree[cur_node].c++;
    return tree[cur_node].c;
  }

};
```

4 Dynamic Programming Optimization

4.1 Convex Hull Trick

```
#define long long long
#define pll pair <long, long>
#define all(c) c.begin(), c.end()
#define fastio ios_base::sync_with_stdio(false);
    cin.tie(0)

struct line{
    long a, b;
    line() {};
    line(long a, long b) : a(a), b(b) {};
    bool operator < (const line &A) const {
            return pll(a,b) < pll(A.a,A.b);
        }
};

bool bad(line A, line B, line C){
    return (C.b - B.b) * (A.a - B.a) <= (B.b -
        A.b) * (B.a - C.a);
}

void addLine(vector<line> &memo, line cur){
    int k = memo.size();
    while (k >= 2 && bad(memo[k - 2], memo[k -
        1], cur)){
        memo.pop_back();
        k--;
    }
    memo.push_back(cur);
}
long Fn(line A, long x){
    return A.a * x + A.b;
}

long query(vector<line> &memo, long x){
    int lo = 0, hi = memo.size() - 1;
    while (lo != hi){
        int mi = (lo + hi) / 2;
        if (Fn(memo[mi], x) > Fn(memo[mi + 1], x)){
            lo = mi + 1;
        }
        else hi = mi;
    }
    return Fn(memo[lo], x);
}

const int N = 1e6 + 1;
long dp[N];

int main()
{
    fastio;
    int n, c; cin >> n >> c;
    vector<line> memo;
    for (int i = 1; i <= n; i++){
        long val; cin >> val;
        addLine(memo, {-2 * val, val * val + dp[i
            - 1]});
        dp[i] = query(memo, val) + val * val + c;
    }
    cout << dp[n] << '\n';
    return 0;
}
```

4.2 Divide and Conquer

```
/**
 * recurrence:
 *    dp[k][i] = min dp[k-1][j] + c[i][j - 1], for
      all j > i;
 *
 * "comp" computes dp[k][i] for all i in O(n log
      n) (k is fixed)
 *
 * Problems:
 *  https://icpc.kattis.com/problems/branch
 *  http://codeforces.com/contest/321/problem/E
 * */

void comp(int l, int r, int le, int re) {
  if (l > r) return;

  int mid = (l + r) >> 1;

  int best = max(mid + 1, le);
  dp[cur][mid] = dp[cur ^ 1][best] + cost(mid,
      best - 1);
  for (int i = best; i <= re; i++) {
    if (dp[cur][mid] > dp[cur ^ 1][i] + cost(mid,
        i - 1)) {
      best = i;
      dp[cur][mid] = dp[cur ^ 1][i] + cost(mid, i
          - 1);
    }
  }

  comp(l, mid - 1, le, best);
  comp(mid + 1, r, best, re);
}
```