Team notebook

August 30, 2015

Contents

1 Algorithms

1.1 sliding window

```
/*
 * Given an array ARR and an integer K, the problem boils down to
     computing for each index i: min(ARR[i], ARR[i-1], ..., ARR[i-K+1]).
 * if mx == true, returns the maximun.
 * http://people.cs.uct.ac.za/~ksmith/articles/sliding_window_minimum.html
 * */

vector<int> sliding_window_minmax(vector<int> & ARR, int K, bool mx) {
  deque< pair<int, int> > window;
  vector<int> ans;
  for (int i = 0; i < ARR.size(); i++) {
    if (mx) {
      while (!window.empty() && window.back().first <= ARR[i])
        window.pop_back();
    } else {
```

```
      while (!window.empty() && window.back().first >= ARR[i])
        window.pop_back();
    }
    window.push_back(make_pair(ARR[i], i));

    while(window.front().second <= i - K)
      window.pop_front();

    ans.push_back(window.front().first);
  }
  return ans;
}
```

2 Data structures

2.1 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));
  }
};
```

2.2 heavy light decomposition

```
// Heavy-Light Decomposition
struct TreeDecomposition {
  vector<int> g[MAXN], c[MAXN];
int s[MAXN]; // subtree size
int p[MAXN]; // parent id
int r[MAXN]; // chain root id
int t[MAXN]; // index used in segtree/bit/...
int d[MAXN]; // depht
int ts;

void dfs(int v, int f) {
  p[v] = f;
  s[v] = 1;
  if (f != -1) d[v] = d[f] + 1;
  else d[v] = 0;

  for (int i = 0; i < g[v].size(); ++i) {
    int w = g[v][i];
    if (w != f) {
      dfs(w, v);
      s[v] += s[w];
    }
  }
}

void hld(int v, int f, int k) {
  t[v] = ts++;
  c[k].push_back(v);
  r[v] = k;

  int x = 0, y = -1;
  for (int i = 0; i < g[v].size(); ++i) {
    int w = g[v][i];
    if (w != f) {
      if (s[w] > x) {
        x = s[w];
        y = w;
      }
    }
  }
  if (y != -1) {
    hld(y, v, k);
  }

  for (int i = 0; i < g[v].size(); ++i) {
    int w = g[v][i];
    if (w != f && w != y) {
      hld(w, v, w);
```

```
      }
    }
  }

  void init(int n) {
    for (int i = 0; i < n; ++i) {
      g[i].clear();
    }
  }

  void add(int a, int b) {
    g[a].push_back(b);
    g[b].push_back(a);
  }

  void build() {
    ts = 0;
    dfs(0, -1);
    hld(0, 0, 0);
  }
};
```

2.3 segment tree

```
/**
 * Taken from: http://codeforces.com/blog/entry/18051
 * */

const int N = 1e5; // limit for array size
int n; // array size
int t[2 * N];

void build() { // build the tree
  for (int i = n - 1; i > 0; --i) t[i] = t[i<<1] + t[i<<1|1];
}

// Single modification, range query.
void modify(int p, int value) { // set value at position p
  for (t[p += n] = value; p > 1; p >>= 1) t[p>>1] = t[p] + t[p^1];
}

int query(int l, int r) { // sum on interval [l, r)
  int res = 0;
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l&1) res += t[l++];
    if (r&1) res += t[--r];
  }
  return res;
}

// Range modification, single query.

void modify(int l, int r, int value) {
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l&1) t[l++] += value;
    if (r&1) t[--r] += value;
  }
}

int query(int p) {
  int res = 0;
  for (p += n; p > 0; p >>= 1) res += t[p];
  return res;
}

/**
 * If at some point after modifications we need to inspect all the
 * elements in the array, we can push all the modifications to the
 * leaves using the following code. After that we can just traverse
 * elements starting with index n. This way we reduce the complexity
 * from O(n log(n)) to O(n) similarly to using build instead of n
     modifications.
 * */

void push() {
  for (int i = 1; i < n; ++i) {
    t[i<<1] += t[i];
    t[i<<1|1] += t[i];
    t[i] = 0;
  }
}

// Non commutative combiner functions.

void modify(int p, const S& value) {
  for (t[p += n] = value; p >>= 1; ) t[p] = combine(t[p<<1], t[p<<1|1]);
}
```

```
S query(int l, int r) {
  S resl, resr;
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l&1) resl = combine(resl, t[l++]);
    if (r&1) resr = combine(t[--r], resr);
  }
  return combine(resl, resr);
}

// To be continued ...
```

2.4 sparse table

```
// RMQ.
const int MN = 100000 + 10; // Max number of elements
const int ML = 18; // ceil(log2(MN));

struct st {
  int data[MN];
  int M[MN][ML];
  int n;

  void read(int _n) {
    n = _n;
    for (int i = 0; i < n; ++i)
      cin >> data[i];
  }

  void build() {
    for (int i = 0; i < n; ++i)
      M[i][0] = data[i];
    for (int j = 1, p = 2, q = 1; p <= n; ++j, p <<= 1, q <<= 1)
      for (int i = 0; i + p - 1 < n; ++i)
        M[i][j] = max(M[i][j - 1], M[i + q][j - 1]);
  }
  int query(int b, int e) {
    int k = log2(e - b + 1);
    return max(M[b][k], M[e + 1 - (1<<k)][k]);
  }
};
```

2.5 splay tree

```
using namespace std;
#include<bits/stdc++.h>
#define D(x) cout<<x<<endl;

typedef int T;

struct node{
  node *left, *right, *parent;
  T key;
  node (T k) : key(k), left(0), right(0), parent(0) {}
};

struct splay_tree{


  node *root;

  void right_rot(node *x) {
    node *p = x->parent;
    if (x->parent = p->parent) {
      if (x->parent->left == p) x->parent->left = x;
      if (x->parent->right == p) x->parent->right = x;
    }
    if (p->left = x->right) p->left->parent = p;
    x->right = p;
    p->parent = x;
  }

  void left_rot(node *x) {
    node *p = x->parent;
    if (x->parent = p->parent) {
      if (x->parent->left == p) x->parent->left = x;
      if (x->parent->right == p) x->parent->right = x;
    }
    if (p->right = x->left) p->right->parent = p;
    x->left = p;
    p->parent = x;
  }

  void splay(node *x, node *fa = 0) {

    while( x->parent != fa and x->parent != 0) {
      node *p = x->parent;
```

```
    if (p->parent == fa)
      if (p->right == x)
        left_rot(x);
      else
        right_rot(x);
    else {
      node *gp = p->parent; //grand parent
      if (gp->left == p)
        if (p->left == x)
          right_rot(x),right_rot(x);
        else
          left_rot(x),right_rot(x);
      else
        if (p->left == x)
          right_rot(x), left_rot(x);
        else
          left_rot(x), left_rot(x);
    }
  }
  if (fa == 0) root = x;
}

void insert(T key) {
  node *cur = root;
  node *pcur = 0;
  while (cur) {
    pcur = cur;
    if (key > cur->key) cur = cur->right;
    else cur = cur->left;
  }
  cur = new node(key);
  cur->parent = pcur;
  if (!pcur) root = cur;
  else if (key > pcur->key ) pcur->right = cur;
  else pcur->left = cur;
  splay(cur);
}

node *find(T key) {
  node *cur = root;
  while (cur) {
    if (key > cur->key) cur = cur->right;
    else if(key < cur->key) cur = cur->left;
    else return cur;
  }
    return 0;
  }

  splay_tree(){ root = 0;};
};
```

2.6 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;
  }

};
```

3 Graphs

3.1 directed mst

```
const int inf = 1000000 + 10;

struct edge {
  int u, v, w;
  edge() {}
  edge(int a,int b,int c) : u(a), v(b), w(c) {}
};

/**
 * Computes the minimum spanning tree for a directed graph
 * - edges : Graph description in the form of list of edges.
 *    each edge is: From node u to node v with cost w
 * - root : Id of the node to start the DMST.
 * - n    : Number of nodes in the graph.
 * */

int dmst(vector<edge> &edges, int root, int n) {
  int ans = 0;
  int cur_nodes = n;
  while (true) {
    vector<int> lo(cur_nodes, inf), pi(cur_nodes, inf);
    for (int i = 0; i < edges.size(); ++i) {
      int u = edges[i].u, v = edges[i].v, w = edges[i].w;
      if (w < lo[v] and u != v) {
        lo[v] = w;
        pi[v] = u;
      }
    }

    lo[root] = 0;
    for (int i = 0; i < lo.size(); ++i) {
      if (i == root) continue;
      if (lo[i] == inf) return -1;
    }
    int cur_id = 0;
    vector<int> id(cur_nodes, -1), mark(cur_nodes, -1);
    for (int i = 0; i < cur_nodes; ++i) {
      ans += lo[i];
      int u = i;
      while (u != root and id[u] < 0 and mark[u] != i) {
        mark[u] = i;
        u = pi[u];
      }
      if (u != root and id[u] < 0) { // Cycle
        for (int v = pi[u]; v != u; v = pi[v])
          id[v] = cur_id;
        id[u] = cur_id++;
      }
    }

    if (cur_id == 0)
      break;

    for (int i = 0; i < cur_nodes; ++i)
      if (id[i] < 0) id[i] = cur_id++;

    for (int i = 0; i < edges.size(); ++i) {
      int u = edges[i].u, v = edges[i].v, w = edges[i].w;
      edges[i].u = id[u];
      edges[i].v = id[v];
      if (id[u] != id[v])
        edges[i].w -= lo[v];
    }
    cur_nodes = cur_id;
    root = id[root];
  }

  return ans;
}
```

3.2 konig's theorem

In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover

3.3 minimum path cover in DAG

Given a directed acyclic graph $G = (V, E)$, we are to find the minimum number of vertex-disjoint paths to cover each vertex in V.

We can construct a bipartite graph $G' = (Vout \cup Vin, E')$ from G, where :

$$Vout = \{v \in V : v\ has\ positive\ out-degree\}$$

$$Vin = \{v \in V : v\ has\ positive\ in-degree\}$$

$$E' = \{(u, v) \in Vout \times Vin : (u, v) \in E\}$$

Then it can be shown, via König's theorem, that G' has a matching of size m if and only if there exists $n - m$ vertex-disjoint paths that cover each vertex in G, where n is the number of vertices in G and m is the maximum cardinality bipartite mathching in G'.

Therefore, the problem can be solved by finding the maximum cardinality matching in G' instead.

NOTE: If the paths are note necesarily disjoints, find the transitive closure and solve the problem for disjoint paths.

3.4 tarjan scc

```
const int MN = 20002;

struct tarjan_scc {
  int scc[MN], low[MN], d[MN], stacked[MN];
  int ticks, current_scc;
  deque<int> s; // used as stack.

  tarjan_scc() {}

  void init () {
    memset(scc, -1, sizeof scc);
    memset(d, -1, sizeof d);
    memset(stacked, 0, sizeof stacked);
    s.clear();
    ticks = current_scc = 0;
  }

  void compute(vector<vector<int> > &g, int u) {
    d[u] = low[u] = ticks++;
    s.push_back(u);
    stacked[u] = true;
    for (int i = 0; i < g[u].size(); ++i) {
      int v = g[u][i];
      if (d[v] == -1)
        compute(g, v);
      if (stacked[v]) {
        low[u] = min(low[u], low[v]);
      }
    }

    if (d[u] == low[u]) { // root
      int v;
      do {
        v = s.back();s.pop_back();
        stacked[v] = false;
        scc[v] = current_scc;
      } while (u != v);
      current_scc++;
    }
  }
};
```

4 Matrix

4.1 matrix

```
const int MN = 111;
const int mod = 10000;

struct matrix {
  int r, c;
  int m[MN][MN];

  matrix (int _r, int _c) : r (_r), c (_c) {
    memset(m, 0, sizeof m);
  }

  void print() {
    for (int i = 0; i < r; ++i) {
      for (int j = 0; j < c; ++j)
        cout << m[i][j] << " ";
      cout << endl;
```

```
    }
  }

  int x[MN][MN];
  matrix & operator *= (const matrix &o) {
    memset(x, 0, sizeof x);
    for (int i = 0; i < r; ++i)
      for (int k = 0; k < c; ++k)
        if (m[i][k] != 0)
          for (int j = 0; j < c; ++j) {
            x[i][j] = (x[i][j] + ((m[i][k] * o.m[k][j]) % mod) ) % mod;
          }
    memcpy(m, x, sizeof(m));
    return *this;
  }
};

void matrix_pow(matrix b, long long e, matrix &res) {
  memset(res.m, 0, sizeof res.m);
  for (int i = 0; i < b.r; ++i)
    res.m[i][i] = 1;

  if (e == 0) return;
  while (true) {
    if (e & 1) res *= b;
    if ((e >>= 1) == 0) break;
    b *= b;
  }
}
```

5 Misc

5.1 Template Java

```
import java.io.*;
import java.util.StringTokenizer;

public class Template {

    public static void main(String []args) throws IOException {
        Scanner in = new Scanner(System.in);
        OutputWriter out = new OutputWriter(System.out);
        Task solver = new Task();
        solver.solve(in, out);
        out.close();
    }

}

class Task{
    public void solve(Scanner in, OutputWriter out){

    }
}

class Scanner{
    public BufferedReader reader;
    public StringTokenizer st;

    public Scanner(InputStream stream){
        reader = new BufferedReader(new InputStreamReader(stream));
        st = null;
    }

    public String next(){
        while(st == null || !st.hasMoreTokens()){
            try{
                String line = reader.readLine();
                if(line == null) return null;
                st = new StringTokenizer(line);
            }catch (Exception e){
                throw (new RuntimeException());
            }
        }
        return st.nextToken();
    }

    public int nextInt(){
        return Integer.parseInt(next());
    }
    public long nextLong(){
        return Long.parseLong(next());
    }
    public double nextDouble(){
        return Double.parseDouble(next());
    }
}
```

```
class OutputWriter{
    BufferedWriter writer;

    public OutputWriter(OutputStream stream){
        writer = new BufferedWriter(new OutputStreamWriter(stream));
    }

    public void print(int i) throws IOException {
        writer.write(i);
    }

    public void print(String s) throws IOException {
        writer.write(s);
    }

    public void print(char []c) throws IOException {
        writer.write(c);
    }
    public void close() throws IOException {
        writer.close();
    }

}
```

5.2 io

```
// taken from :
    https://github.com/lbv/pc-code/blob/master/solved/c-e/diablo/diablo.cpp
// this is very fast as well :
    https://github.com/lbv/pc-code/blob/master/code/input.cpp

typedef unsigned int u32;
#define BUF 524288
struct Reader {
  char buf[BUF]; char b; int bi, bz;
  Reader() { bi=bz=0; read(); }
  void read() {
    if (bi==bz) { bi=0; bz = fread(buf, 1, BUF, stdin); }
    b = bz ? buf[bi++] : 0; }
  void skip() { while (b > 0 && b <= 32) read(); }
  u32 next_u32() {
    u32 v = 0; for (skip(); b > 32; read()) v = v*10 + b-48; return v; }
  int next_int() {
    int v = 0; bool s = false;
    skip(); if (b == '-') { s = true; read(); }
    for (; 48<=b&&b<=57; read()) v = v*10 + b-48; return s ? -v : v; }
  char next_char() { skip(); char c = b; read(); return c; }
};
```

6 Number theory

6.1 convolution

```
typedef long long int LL;
typedef pair<LL, LL> PLL;

inline bool is_pow2(LL x) {
  return (x & (x-1)) == 0;
}

inline int ceil_log2(LL x) {
  int ans = 0;
  --x;
  while (x != 0) {
    x >>= 1;
    ans++;
  }
  return ans;
}

/* Returns the convolution of the two given vectors in time proportional
    to n*log(n).
 * The number of roots of unity to use nroots_unity must be set so that
     the product of the first
 * nroots_unity primes of the vector nth_roots_unity is greater than the
     maximum value of the
 * convolution. Never use sizes of vectors bigger than 2^24, if you need
     to change the values of
 * the nth roots of unity to appropriate primes for those sizes.
 */
vector<LL> convolve(const vector<LL> &a, const vector<LL> &b, int
    nroots_unity = 2) {
  int N = 1 << ceil_log2(a.size() + b.size());
  vector<LL> ans(N,0), fA(N), fB(N), fC(N);
```

```
  LL modulo = 1;
  for (int times = 0; times < nroots_unity; times++) {
    fill(fA.begin(), fA.end(), 0);
    fill(fB.begin(), fB.end(), 0);
    for (int i = 0; i < a.size(); i++) fA[i] = a[i];
    for (int i = 0; i < b.size(); i++) fB[i] = b[i];
    LL prime = nth_roots_unity[times].first;
    LL inv_modulo = mod_inv(modulo % prime, prime);
    LL normalize = mod_inv(N, prime);
    ntfft(fA, 1, nth_roots_unity[times]);
    ntfft(fB, 1, nth_roots_unity[times]);
    for (int i = 0; i < N; i++) fC[i] = (fA[i] * fB[i]) % prime;
    ntfft(fC, -1, nth_roots_unity[times]);
    for (int i = 0; i < N; i++) {
      LL curr = (fC[i] * normalize) % prime;
      LL k = (curr - (ans[i] % prime) + prime) % prime;
      k = (k * inv_modulo) % prime;
      ans[i] += modulo * k;
    }
    modulo *= prime;
  }
  return ans;
}
```

6.2 crt

```
/**
 * Chinese remainder theorem.
 * Find z such that z % x[i] = a[i] for all i.
 * */
long long crt(vector<long long> &a, vector<long long> &x) {
  long long z = 0;
  long long n = 1;
  for (int i = 0; i < x.size(); ++i)
    n *= x[i];

  for (int i = 0; i < a.size(); ++i) {
    long long tmp = (a[i] * (n / x[i])) % n;
    tmp = (tmp * mod_inv(n / x[i], x[i])) % n;
    z = (z + tmp) % n;
  }

  return (z + n) % n;
}
```

6.3 discrete logarithm

```
// Computes x which a ^ x = b mod n.

long long d_log(long long a, long long b, long long n) {
  long long m = ceil(sqrt(n));
  long long aj = 1;
  map<long long, long long> M;
  for (int i = 0; i < m; ++i) {
    if (!M.count(aj))
      M[aj] = i;
    aj = (aj * a) % n;
  }

  long long coef = mod_pow(a, n - 2, n);
  coef = mod_pow(coef, m, n);
  // coef = a ^ (-m)
  long long gamma = b;
  for (int i = 0; i < m; ++i) {
    if (M.count(gamma)) {
      return i * m + M[gamma];
    } else {
      gamma = (gamma * coef) % n;
    }
  }
  return -1;
}
```

6.4 ext euclidean

```
void ext_euclid(long long a, long long b, long long &x, long long &y,
    long long &g) {
  x = 0, y = 1, g = b;
  long long m, n, q, r;
  for (long long u = 1, v = 0; a != 0; g = a, a = r) {
    q = g / a, r = g % a;
    m = x - u * q, n = y - v * q;
    x = u, y = v, u = m, v = n;
```

```
  }
}
```

6.5 highest exponent factorial

```
int highest_exponent(int p, const int &n){
  int ans = 0;
  int t = p;
  while(t <= n){
    ans += n/t;
    t*=p;
  }
  return ans;
}
```

6.6 miller rabin

```
const int rounds = 20;

// checks whether a is a witness that n is not prime, 1 < a < n
bool witness(long long a, long long n) {
  // check as in Miller Rabin Primality Test described
  long long u = n - 1;
  int t = 0;
  while (u % 2 == 0) {
    t++;
    u >>= 1;
  }
  long long next = mod_pow(a, u, n);
  if (next == 1) return false;
  long long last;
  for (int i = 0; i < t; ++i) {
    last = next;
    next = mod_mul(last, last, n);
    if (next == 1) {
      return last != n - 1;
    }
  }
  return next != 1;
}

// Checks if a number is prime with prob 1 - 1 / (2 ^ it)
//  D(miller_rabin(999999999999999997LL) == 1);
//  D(miller_rabin(9999999999971LL) == 1);
//  D(miller_rabin(7907) == 1);
bool miller_rabin(long long n, int it = rounds) {
  if (n <= 1) return false;
  if (n == 2) return true;
  if (n % 2 == 0) return false;
  for (int i = 0; i < it; ++i) {
    long long a = rand() % (n - 1) + 1;
    if (witness(a, n)) {
      return false;
    }
  }
  return true;
}
```

6.7 mod inv

```
long long mod_inv(long long n, long long m) {
  long long x, y, gcd;
  ext_euclid(n, m, x, y, gcd);
  if (gcd != 1)
    return 0;
  return (x + m) % m;
}
```

6.8 mod mul

```
// Computes (a * b) % mod
long long mod_mul(long long a, long long b, long long mod) {
  long long x = 0, y = a % mod;
  while (b > 0) {
    if (b & 1)
      x = (x + y) % mod;
    y = (y * 2) % mod;
    b /= 2;
  }
  return x % mod;
}
```

6.9 mod pow

```
// Computes ( a ^ exp ) % mod.
long long mod_pow(long long a, long long exp, long long mod) {
  long long ans = 1;
  while (exp > 0) {
    if (exp & 1)
      ans = mod_mul(ans, a, mod);
    a = mod_mul(a, a, mod);
    exp >>= 1;
  }
  return ans;
}
```

6.10 number theoretic transform

```
typedef long long int LL;
typedef pair<LL, LL> PLL;

/* The following vector of pairs contains pairs (prime, generator)
 * where the prime has an Nth root of unity for N being a power of two.
 * The generator is a number g s.t g^(p-1)=1 (mod p)
 * but is different from 1 for all smaller powers */
vector<PLL> nth_roots_unity {
  {1224736769,330732430},{1711276033,927759239},{167772161,167489322},
   {469762049,343261969},{754974721,643797295},{1107296257,883865065}};

PLL ext_euclid(LL a, LL b) {
  if (b == 0)
    return make_pair(1,0);
  pair<LL,LL> rc = ext_euclid(b, a % b);
  return make_pair(rc.second, rc.first - (a / b) * rc.second);
}

//returns -1 if there is no unique modular inverse
LL mod_inv(LL x, LL modulo) {
  PLL p = ext_euclid(x, modulo);
  if ( (p.first * x + p.second * modulo) != 1 )
    return -1;
  return (p.first+modulo) % modulo;
}

//Number theory fft. The size of a must be a power of 2
void ntfft(vector<LL> &a, int dir, const PLL &root_unity) {
  int n = a.size();
  LL prime = root_unity.first;
  LL basew = mod_pow(root_unity.second, (prime-1) / n, prime);
  if (dir < 0) basew = mod_inv(basew, prime);
  for (int m = n; m >= 2; m >>= 1) {
    int mh = m >> 1;
    LL w = 1;
    for (int i = 0; i < mh; i++) {
      for (int j = i; j < n; j += m) {
        int k = j + mh;
        LL x = (a[j] - a[k] + prime) % prime;
        a[j] = (a[j] + a[k]) % prime;
        a[k] = (w * x) % prime;
      }
      w = (w * basew) % prime;
    }
    basew = (basew * basew) % prime;
  }
  int i = 0;
  for (int j = 1; j < n - 1; j++) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j < i) swap(a[i], a[j]);
  }
}
```

6.11 pollard rho factorize

```
long long pollard_rho(long long n) {
  long long x, y, i = 1, k = 2, d;
  x = y = rand() % n;
  while (1) {
    ++i;
    x = mod_mul(x, x, n);
    x += 2;
    if (x >= n) x -= n;
    if (x == y) return 1;
    d = __gcd(abs(x - y), n);
    if (d != 1) return d;
    if (i == k) {
      y = x;
      k *= 2;
```

```
    }
  }
  return 1;
}


// Returns a list with the prime divisors of n
vector<long long> factorize(long long n) {
  vector<long long> ans;
  if (n == 1)
    return ans;
  if (miller_rabin(n)) {
    ans.push_back(n);
  } else {
    long long d = 1;
    while (d == 1)
      d = pollard_rho(n);
    vector<long long> dd = factorize(d);
    ans = factorize(n / d);
    for (int i = 0; i < dd.size(); ++i)
      ans.push_back(dd[i]);
  }
  return ans;
}
```

7 Strings

7.1 minimal string rotation

```
// Lexicographically minimal string rotation
int lmsr() {
  string s;
  cin >> s;
  int n = s.size();
  s += s;
  vector<int> f(s.size(), -1);
  int k = 0;
  for (int j = 1; j < 2 * n; ++j) {
    int i = f[j - k - 1];
    while (i != -1 && s[j] != s[k + i + 1]) {
      if (s[j] < s[k + i + 1])
        k = j - i - 1;
      i = f[i];
    }
    if (i == -1 && s[j] != s[k + i + 1]) {
      if (s[j] < s[k + i + 1]) {
        k = j;
      }
      f[j - k] = -1;
    } else {
      f[j - k] = i + 1;
    }
  }
  return k;
}
```

7.2 suffix array

```
/**
 * O (n log^2 (n))
 * See http://web.stanford.edu/class/cs97si/suffix-array.pdf for reference
 * */
using namespace std;
#include<bits/stdc++.h>
#define D(x) cout<<#x " = "<<(x)<<endl

struct entry{
  int a, b, p;
  entry(){}
  entry(int x, int y, int z): a(x), b(y), p(z){}
  bool operator < (const entry &o) const {
    return (a == o.a) ? (b < o.b) : (a < o.a);
  }
};

struct SuffixArray{
  const int N;
  string s;
  vector<vector<int> > P;
  vector<entry> M;

  SuffixArray(const string &s) : N(s.length()) , s(s), P(1, vector<int>
      (N, 0)), M(N) {
    for (int i = 0; i < N; ++i)
      P[0][i] = s[i];
```

```
    for (int skip = 1, level = 1; skip < N; skip *= 2, level++) {
      P.push_back(vector<int>(N, 0));
      for (int i = 0 ; i < N; ++i) {
        int next = ((i + skip) < N) ? P[level - 1][i + skip] : -10000;
        M[i] = entry(P[level - 1][i], next, i);
      }
      sort(M.begin(), M.end());
      for (int i = 0; i < N; ++i)
        P[level][M[i].p] = (i > 0 and M[i].a == M[i - 1].a and M[i].b ==
             M[i - 1].b) ? P[level - 1][M[i - 1].p] : i;
    }
  }

  vector<int> getSuffixArray(){
    return P.back();
  }

  // returns the length of the longest common prefix of s[i...L-1] and
      s[j...L-1]
  int longestCommonPrefix(int i, int j) {
    int len = 0;
    if (i == j) return N - i;
    for (int k = P.size() - 1; k >= 0 && i < N && j < N; --k) {
      if (P[k][i] == P[k][j]) {
        i += 1 << k;
        j += 1 << k;
        len += 1 << k;
      }
    }
    return len;
  }
};
```

7.3 z algorithm

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

vector<int> compute_z(const string &s){
  int n = s.size();
  vector<int> z(n,0);
  int l,r;
  r = l = 0;
  for(int i = 1; i < n; ++i){
    if(i > r) {
      l = r = i;
      while(r < n and s[r - l] == s[r])r++;
      z[i] = r - l;r--;
    }else{
      int k = i-l;
      if(z[k] < r - i +1) z[i] = z[k];
      else {
        l = i;
        while(r < n and s[r - l] == s[r])r++;
        z[i] = r - l;r--;
      }
    }
  }
  return z;
}

int main(){

  //string line;cin>>line;
  string line = "alfalfa";
  vector<int> z = compute_z(line);

  for(int i = 0; i < z.size(); ++i ){
    if(i)cout<<" ";
    cout<<z[i];
  }
  cout<<endl;

  // must print "0 0 0 4 0 0 1"

  return 0;
}
```