Sharif University of Technology - Kolompeh - Notebook

Contents

1 Geometry 1
- 1.1 Rotating Calipers . . . 1
- 1.2 Delaunay Triangulation O(n2) . . . 1

2 String 3
- 2.1 Suffix Automata . . . 3
- 2.2 Suffix Tree . . . 3
- 2.3 Palindromic Tree . . . 3

3 Data structure 4
- 3.1 Treap . . . 4
- 3.2 Link-cut Tree . . . 4
- 3.3 Dynimic convex hull . . . 5

4 Graph 6
- 4.1 Maximum matching - Edmond's blossom . . . 6
- 4.2 Biconnected components . . . 7
- 4.3 Maximum weighted matching - Hungarian . . . 7
- 4.4 Ear decomposition . . . 8
- 4.5 Stoer-Wagner min cut $O(n3)$ . . . 8
- 4.6 Directed minimum spanning tree $O(m \log n)$ . . . 8
- 4.7 Directed minimum spanning tree $O(nm)$ . . . 9
- 4.8 Dominator tree . . . 10

5 Combinatorics 11
- 5.1 LP simplex . . . 11
- 5.2 FFT . . . 12
- 5.3 NTT . . . 12
- 5.4 Base Vectors in Z2 . . . 13

6 !]gauss.cpp Gaussian Eliminatio 13
- 6.1 Stirling 1 . . . 13
- 6.2 Stirling 2 . . . 14
- 6.3 Chinese remainder . . . 14
- 6.4 Popular LP . . . 15
- 6.5 Duality of LP . . . 15
- 6.6 Extended catalan . . . 15
- 6.7 Find polynomial from it's points . . . 15

7 Constants 15
- 7.1 Number of primes . . . 15
- 7.2 Factorials . . . 15
- 7.3 Powers of 3 . . . 15
- 7.4 C(2n,n) . . . 16
- 7.5 Most divisor . . . 16

1 Geometry

1.1 Rotating Calipers

```
vector <pair<pt, pt>> get_antipodals(vector <pt> &p){
        int n = sz(p);
        sort(p.begin(), p.end());
        vector <pt> U, L;
        for (int i = 0; i < n; i++){
                while (sz(U) > 1 && side(U[sz(U)-2], U[sz(U)-1], p[i])
                     >= 0)
                        U.pop_back();
                while (sz(L) > 1 && side(L[sz(L)-2], L[sz(L)-1], p[i])
                     <= 0)
                        L.pop_back();
                U.pb(p[i]);
                L.pb(p[i]);
        }
        vector <pair<pt, pt>> res;
        int i = 0, j = sz(L)-1;
        while (i+1 < sz(U) || j > 0){
                res.pb({U[i], L[j]});
                if (i+1 == sz(U))
                        j--;
                else if (j == 0)
                        i++;
                else if (cross(L[j]-L[j-1], U[i+1]-U[i]) >= 0)
                        i++;
                else
                        j--;
        }
        return res;
}
```

1.2 Delaunay Triangulation O(n2)

```
struct Delaunay{
        vector <pt> p;
        vector <pt> to;
        vector <int> nxt;

        int add_edge(pt q, int bef=-1){
                int cnt = sz(to);
                to.pb(q);
                nxt.pb(-1);
                if (bef != -1){
                        nxt[bef] = cnt;
                        to.pb(to[bef]);
                        nxt.pb(-1);
                }
                return cnt;
        }

        int before(int e){
                int cur = e, last = -1;
                do{
                        last = cur;
                        cur = nxt[cur^1];
        }while (cur != e);
        return last^1;
}

void easy_triangulate(){
        to.clear();
        nxt.clear();
        sort(p.begin(), p.end());
        if (dir(p[0], p[1], p[2]) > 0)
                swap(p[1], p[2]);
        int to0 = add_edge(p[0]), to0c = add_edge(p[2]),
                to1 = add_edge(p[1]), to1c = add_edge(p[0]),
                to2 = add_edge(p[2]), to2c = add_edge(p[1]);

        nxt[to1] = to2; nxt[to2] = to0;
        nxt[to0] = to1; nxt[to0c] = to2c;
        nxt[to2c] = to1c; nxt[to1c] = to0c;

        int e = to0;
        for (int i = 3; i < sz(p); i++){
                pt q = p[i];
                while (dir(q, to[e^1], to[e]) < 0 || dir(q, to
                    [e^1], to[before(e)^1]) < 0)
                        e = nxt[e];
                vector <int> vis;
                while (dir(q, to[e^1], to[e]) > 0 || dir(q, to
                    [e^1], to[before(e)^1]) > 0){
                        vis.pb(e);
                        e = nxt[e];
                }
                int ex = add_edge(q, before(vis[0]));
                int last = ex^1;
                for (int edge : vis){
                        nxt[last] = edge;
                        int eq = add_edge(q, edge);
                        nxt[edge] = eq;
                        nxt[eq] = last;
                        last = eq^1;
                }
                nxt[ex] = last;
                nxt[last] = e;
        }
}

bool incircle(pt a, pt b, pt c, pt d) {
        return a.z() * (b.x * (c.y - d.y) - c.x * (b.y - d.y)
            + d.x * (b.y - c.y))
                - b.z() * (a.x * (c.y - d.y) - c.x * (a.y - d.
                    y) + d.x * (a.y - c.y))
                + c.z() * (a.x * (b.y - d.y) - b.x * (a.y - d.
                    y) + d.x * (a.y - b.y))
                - d.z() * (a.x * (b.y - c.y) - b.x * (a.y - c.
                    y) + c.x * (a.y - b.y)) > 0;
}


bool locally(int e){
        pt a = to[e^1], b = to[e], c = to[nxt[e]], d = to[nxt[
            e^1]];
        if (dir(a, b, c) < 0) return true;
        if (dir(b, a, d) < 0) return true;
        if (incircle(a, b, c, d)) return false;
        if (incircle(b, a, d, c)) return false;
        return true;
}

void flip(int e){
        int a = nxt[e], b = nxt[a],
                c = nxt[e^1], d = nxt[c];
        nxt[d] = a;
        nxt[b] = c;
        to[e] = to[c];
        nxt[a] = e;
        to[e^1] = to[a];
        nxt[c] = e^1;
}

void delaunay_triangulate(){
        if (sz(to) == 0)
                easy_triangulate();
        bool *mark = new bool[sz(to)];
        fill(mark, mark + sz(to), false);
        vector <int> bad;
        for (int e = 0; e < sz(to); e++){
                if (!mark[e/2] && !locally(e)){
                        bad.pb(e);
                        mark[e/2] = true;
                }
        }
        while (sz(bad)){
                int e = bad.back();
                bad.pop_back();
                mark[e/2] = false;
                if (!locally(e)){
                        flip(e);
                        int to_check[4] = {nxt[e], nxt[nxt[e
                            ]], nxt[e^1], nxt[nxt[e^1]]};
                        for (int i = 0; i < 4; i++)
                                if (!mark[to_check[i]/2] && !
                                    locally(to_check[i])){
                                        bad.pb(to_check[i]);
                                        mark[to_check[i]/2] =
                                            true;
                                }
                }
        }
}

vector <tri> get_triangles(){
        vector <tri> res;
        for (int e = 0; e < sz(to); e++){
                pt a = to[e^1], b = to[e], c = to[nxt[e]];
                if (dir(a, b, c) < 0) continue;
                res.pb(tri(a, b, c));
        }
        return res;
}
Delaunay(vector <pt> p):p(p){}
};
```

2 String

2.1 Suffix Automata

```
const int maxn = 2 e5 + 42; // Maximum amount of states
map < char , int > to [ maxn ]; // Transitions
int link [ maxn ]; // Suffix links
int len [ maxn ]; // Lengthes of largest strings in states
int last = 0; // State corresponding to the whole string
int sz = 1; // Current amount of states
void add_letter ( char c ) { // Adding character to the end
    int p = last ; // State of string s
    last = sz ++; // Create state for string sc
    len [ last ] = len [ p ] + 1;
    for (; to [ p ][ c ] == 0; p = link [ p ]) // (1)
        to [ p ][ c ] = last ; // Jumps which add new suffixes
    if ( to [ p ][ c ] == last ) { // This is the first occurrence of
        c if we are here
        link [ last ] = 0;
        return ;
    }
    int q = to [ p ][ c ];
    if ( len [ q ] == len [ p ] + 1) {
        link [ last ] = q ;
        return ;
    }
    // We split off cl from q here
    int cl = sz ++;
    to [ cl ] = to [ q ]; // (2)
    link [ cl ] = link [ q ];
    len [ cl ] = len [ p ] + 1;
    link [ last ] = link [ q ] = cl ;
    for (; to [ p ][ c ] == q ; p = link [ p ]) // (3)
        to [ p ][ c ] = cl ; // Redirect transitions where needed
}
```

2.2 Suffix Tree

```
#define fpos adla
const int inf = 1e9;
const int maxn = 1e4;
char s[maxn];
map<int, int> to[maxn];
int len[maxn], fpos[maxn], link[maxn];
int node, pos;
int sz = 1, n = 0;
int make_node(int _pos, int _len) {
    fpos[sz] = _pos;
    len [sz] = _len;
    return sz++;
}
void go_edge() {
    while(pos > len[to[node][s[n - pos]]]) {
        node = to[node][s[n - pos]];
        pos -= len[node];
    }
}
void add_letter(int c) {
    s[n++] = c;
    pos++;
    int last = 0;
    while(pos > 0) {
        go_edge();
        int edge = s[n - pos];
        int &v = to[node][edge];
        int t = s[fpos[v] + pos - 1];
        if(v == 0) {
            v = make_node(n - pos, inf);
            link[last] = node;
            last = 0;
        } else if(t == c) {
            link[last] = node;
            return;
        } else {
            int u = make_node(fpos[v], pos - 1);
            to[u][c] = make_node(n - 1, inf);
            to[u][t] = v;
            fpos[v] += pos - 1;
            len [v] -= pos - 1;
            v = u;
            link[last] = u;
            last = u;
        }
        if(node == 0)
            pos--;
        else
            node = link[node];
    }
}
int main() {
    len[0] = inf;
    string s;
    cin >> s;
    int ans = 0;
    for(int i = 0; i < s.size(); i++)
        add_letter(s[i]);
    for(int i = 1; i < sz; i++)
        ans += min((int)s.size() - fpos[i], len[i]);
    cout << ans << "\n";
}
```

2.3 Palindromic Tree

```
int n, last, sz;

void init() {
    s[n++] = -1;
    link[0] = 1;
    len[1] = -1;
    sz = 2;
}
int get_link(int v) {
    while(s[n - len[v] - 2] != s[n - 1]) v = link[v];
    return v;
}
void add_letter(int c) {
    s[n++] = c;
    last = get_link(last);
    if(!to[last][c]) {
        len [sz] = len[last] + 2;
        link[sz] = to[get_link(link[last])][c];
        to[last][c] = sz++;
      }
      last = to[last][c];
  }
```

3 Data structure

3.1 Treap

```
struct item {
    int key, prior;
    item * l, * r;
    item() { }
    item (int key, int prior) : key(key), prior(prior), l(NULL), r(
        NULL) { }
};
typedef item * pitem;
void split (pitem t, int key, pitem & l, pitem & r) {
    if (!t)
        l = r = NULL;
    else if (key < t->key)
        split (t->l, key, l, t->l),  r = t;
    else
        split (t->r, key, t->r, r),  l = t;
}
void insert (pitem & t, pitem it) {
    if (!t)
        t = it;
    else if (it->prior > t->prior)
        split (t, it->key, it->l, it->r),  t = it;
    else
        insert (it->key < t->key ? t->l : t->r, it);
}
void merge (pitem & t, pitem l, pitem r) {
    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge (l->r, l->r, r),  t = l;
    else
        merge (r->l, l, r->l),  t = r;
}
void erase (pitem & t, int key) {
    if (t->key == key)
        merge (t, t->l, t->r);
    else
        erase (key < t->key ? t->l : t->r, key);
}
pitem unite (pitem l, pitem r) {
    if (!l || !r)  return l ? l : r;
    if (l->prior < r->prior)  swap (l, r);
    pitem lt, rt;
    split (r, l->key, lt, rt);
    l->l = unite (l->l, lt);
    l->r = unite (l->r, rt);
    return l;
}
```

3.2 Link-cut Tree

```
Node x[N];

struct Node {
    int sz, label; /* size, label */
    Node *p, *pp, *l, *r; /* parent, path-parent, left, right pointers
         */
    Node() { p = pp = l = r = 0; }
};

void update(Node *x) {
    x->sz = 1;
    if(x->l) x->sz += x->l->sz;
    if(x->r) x->sz += x->r->sz;
}

void rotr(Node *x) {
    Node *y, *z;
    y = x->p, z = y->p;
    if((y->l = x->r)) y->l->p = y;
    x->r = y, y->p = x;
    if((x->p = z)) {
        if(y == z->l) z->l = x;
        else z->r = x;
    }
    x->pp = y->pp;
    y->pp = 0;
    update(y);
}

void rotl(Node *x) {
    Node *y, *z;
    y = x->p, z = y->p;
    if((y->r = x->l)) y->r->p = y;
    x->l = y, y->p = x;
    if((x->p = z)) {
        if(y == z->l) z->l = x;
        else z->r = x;
    }
    x->pp = y->pp;
    y->pp = 0;
    update(y);
}

void splay(Node *x) {
    Node *y, *z;
    while(x->p) {
        y = x->p;
        if(y->p == 0) {
            if(x == y->l) rotr(x);
            else rotl(x);
        }
        else {
            z = y->p;
            if(y == z->l) {
                if(x == y->l) rotr(y), rotr(x);
                else rotl(x), rotr(x);
            }
            else {   if(x == y->r) rotl(y), rotl(x);
```

```
                else rotr(x), rotl(x);
            }
        }
    }
    update(x);
}

Node *access(Node *x) {
    splay(x);
    if(x->r) {
        x->r->pp = x;
        x->r->p = 0;
        x->r = 0;
        update(x);
    }

    Node *last = x;
    while(x->pp) {
        Node *y = x->pp;
        last = y;
        splay(y);
        if(y->r) {
            y->r->pp = y;
            y->r->p = 0;
        }
        y->r = x;
        x->p = y;
        x->pp = 0;
        update(y);
        splay(x);
    }
    return last;
}

Node *root(Node *x) {
    access(x);
    while(x->l) x = x->l;
    splay(x);
    return x;
}

void cut(Node *x) {
    access(x);
    x->l->p = 0;
    x->l = 0;
    update(x);
}

void link(Node *x, Node *y) {
    access(x);
    access(y);
    x->l = y;
    y->p = x;
    update(x);
}

Node *lca(Node *x, Node *y) {
    access(x);
    return access(y);
}
```

```
int depth(Node *x) {
    access(x);
    return x->sz - 1;
}

void init(int n) {
    for(int i = 0; i < n; i++) {
        x[i].label = i;
        update(&x[i]);
    }
}
```

3.3 Dynimic convex hull

```
const ld is_query = -(1LL << 62);

struct Line {
    ld m, b;
    mutable std::function<const Line *()> succ;

    bool operator<(const Line &rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line *s = succ();
        if (!s) return 0;
        ld x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};

struct HullDynamic : public multiset<Line> { // dynamic upper hull +
    max value query
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b) * (z->m - y->m) >= (y->b - z->b) * (y->m
            - x->m);
    }

    void insert_line(ld m, ld b) {
        auto y = insert({m, b});
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) {
            erase(y);
            return;
        }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }

    ld best(ld x) {
        auto l = *lower_bound((Line) {x, is_query});
        return l.m * x + l.b;
    }
};
```

4 Graph

4.1 Maximum matching - Edmond's blossom

```
/*
  GETS:
  n->number of vertices
  you should use add_edge(u,v) and
  add pair of vertices as edges (vertices are 0..n-1)
  (note: please don't add multiple edge)

  GIVES:
  output of edmonds() is the maximum matching in general graph
  match[i] is matched pair of i (-1 if there isn't a matched pair)

  O(mn^2)
*/

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

struct struct_edge{int v;struct_edge* nxt;};
typedef struct_edge* edge;
const int MAXN=500;

struct Edmonds
{
  struct_edge pool[MAXN*MAXN*2];
  edge top=pool,adj[MAXN];
  int n,match[MAXN],qh,qt,q[MAXN],father[MAXN],base[MAXN];
  bool inq[MAXN],inb[MAXN];

  void add_edge(int u,int v)
  {
    top->v=v,top->nxt=adj[u],adj[u]=top++;
    top->v=u,top->nxt=adj[v],adj[v]=top++;
  }
  int LCA(int root,int u,int v)
  {
    static bool inp[MAXN];
    memset(inp,0,sizeof(inp));
    while(1)
      {
        inp[u=base[u]]=true;
        if (u==root) break;
        u=father[match[u]];
      }
    while(1)
      {
        if (inp[v=base[v]]) return v;
        else v=father[match[v]];
      }
  }

  void mark_blossom(int lca,int u)
  {
    while (base[u]!=lca)
      {
        int v=match[u];
        inb[base[u]]=inb[base[v]]=true;
        u=father[v];
        if (base[u]!=lca) father[u]=v;
      }
  }

  void blossom_contraction(int s,int u,int v)
  {
    int lca=LCA(s,u,v);
    memset(inb,0,sizeof(inb));
    mark_blossom(lca,u);
    mark_blossom(lca,v);
    if (base[u]!=lca)
      father[u]=v;
    if (base[v]!=lca)
      father[v]=u;
    for (int u=0;u<n;u++)
      if (inb[base[u]])
        {
          base[u]=lca;
          if (!inq[u])
            inq[q[++qt]=u]=true;
        }
  }

  int find_augmenting_path(int s)
  {
    memset(inq,0,sizeof(inq));
    memset(father,-1,sizeof(father));
    for (int i=0;i<n;i++) base[i]=i;
    inq[q[qh=qt=0]=s]=true;
    while (qh<=qt)
      {
        int u=q[qh++];
        for (edge e=adj[u];e;e=e->nxt)
          {
            int v=e->v;
            if (base[u]!=base[v] && match[u]!=v)
              {
                if (v==s || (match[v]!=-1 && father[match[v]]!=-1))
                  blossom_contraction(s,u,v);
                else if (father[v]==-1)
                  {
                    father[v]=u;
                    if (match[v]==-1)
                      return v;
                    else if (!inq[match[v]])
                      inq[q[++qt]=match[v]]=true;
                  }
              }
          }
      }
    return -1;
  }

  int augment_path(int s,int t)
  {
    int u=t,v,w;
    while (u!=-1)
      {
        v=father[u];
        w=match[v];
        match[v]=u;
        match[u]=v;
        u=w;
      }
    return t!=-1;
  }

  int edmonds()
  {
    int matchc=0;
    memset(match,-1,sizeof(match));
    for (int u=0;u<n;u++)
      if (match[u]==-1)
        matchc+=augment_path(u,find_augmenting_path(u));
    return matchc;
  }
};
```

4.2 Biconnected components

```
vector<int> adj[maxn];
bool vis[maxn];
int dep[maxn], par[maxn], lowlink[maxn];
vector<vector<int> > comp;
stack<int> st;
void dfs(int u, int depth = 0, int parent = -1)
{
        vis[u] = true;
        dep[u] = depth;
        par[u] = parent;
        lowlink[u] = depth;
        st.push(u);
        for (int i = 0; i < adj[u].size(); i++)
        {
                int v = adj[u][i];
                if (!vis[v])
                {
                        dfs(v, depth + 1, u);
                        lowlink[u] = min(lowlink[u], lowlink[v]);
                }
                else
                        lowlink[u] = min(lowlink[u], dep[v]);
        }
        if (lowlink[u] == dep[u] - 1)
        {
                comp.push_back(vector<int>());
                while (st.top() != u)
                {
                        comp.back().push_back(st.top());
                        st.pop();
                }
                comp.back().push_back(u);
                st.pop();
                comp.back().push_back(par[u]);
        }
}
void bicon(int n)
{
        for (int i = 0; i < n; i++)
                if (!vis[i])
                        dfs(i);
}
```

4.3 Maximum weighted matching - Hungarian

```
const int N = 2002;
const int INF = 1e9;

int hn, weight[N][N];
int x[N], y[N];

int hungarian()  // maximum weighted perfect matching
{
        int n = hn;
        int p, q;
        vector<int> fx(n, -INF), fy(n, 0);
        fill(x, x + n, -1);
        fill(y, y + n, -1);
        for (int i = 0; i < n; ++i)
                for (int j = 0; j < n; ++j)
                        fx[i] = max(fx[i], weight[i][j]);

        for (int i = 0; i < n; ) {
                vector<int> t(n, -1), s(n+1, i);
                for (p = 0, q = 1; p < q && x[i] < 0; ++p) {
                        int k = s[p];
                        for (int j = 0; j < n && x[i] < 0; ++j)
                                if (fx[k] + fy[j] == weight[k][j] && t
                                    [j] < 0) {
                                        s[q++] = y[j], t[j] = k;
                                        if (y[j] < 0) // match found!
                                                for (int p = j; p >=
                                                    0; j = p)
                                                        y[j] = k = t[j
                                                            ], p = x[k
                                                            ], x[k] =
                                                            j;
                                }
                }
                if (x[i] < 0) {
                        int d = INF;
                        for (int k = 0; k < q; ++k)
                                for (int j = 0; j < n; ++j)
                                        if (t[j] < 0) d = min(d, fx [s
                                            [k]] + fy[j] - weight[s[k
                                            ]][j]);
                        for (int j = 0; j < n; ++j) fy[j] += (t[j] <
                            0? 0: d);
                        for (int k = 0; k < q; ++k) fx[s[k]] -= d;
                } else ++i;
        }
        int ret = 0;
        for (int i = 0; i < n; ++i) ret += weight[i][x[i]];
        return ret;
}
```

4.4 Ear decomposition

```
1- Find a spanning tree of the given graph and choose a root for the
    tree.
2- Determine, for each edge uv that is not part of the tree, the
    distance between the root and the lowest common ancestor of u and
    v.
3- For each edge uv that is part of the tree, find the corresponding "
    master edge", a non-tree edge wx such that the cycle formed by
    adding wx to the tree passes through uv and such that, among such
    edges, w and x have a lowest common ancestor that is as close to
    the root as possible (with ties broken by edge identifiers).
4- Form an ear for each non-tree edge, consisting of it and the tree
    edges for which it is the master, and order the ears by their
    master edges' distance from the root (with the same tie-breaking
    rule).
```

4.5 Stoer-Wagner min cut $O(n3)$

```
const int N = -1, MAXW = -1;

int g[N][N], v[N], w[N], na[N];
bool a[N];

int minCut( int n )
{
    for( int i = 0; i < n; i++ ) v[i] = i;

    int best = MAXW * n * n;
    while( n > 1 )
    {
        // initialize the set A and vertex weights
        a[v[0]] = true;
        for( int i = 1; i < n; i++ )
        {
            a[v[i]] = false;
            na[i - 1] = i;
            w[i] = g[v[0]][v[i]];
        }

        // add the other vertices
        int prev = v[0];
        for( int i = 1; i < n; i++ )
        {
            // find the most tightly connected non-A vertex
            int zj = -1;
            for( int j = 1; j < n; j++ )
                if( !a[v[j]] && ( zj < 0 || w[j] > w[zj] ) )
                    zj = j;

            // add it to A
            a[v[zj]] = true;

            // last vertex?
            if( i == n - 1 )
            {
                // remember the cut weight
                best = min(best, w[zj]);

                // merge prev and v[zj]
                for( int j = 0; j < n; j++ )
                    g[v[j]][prev] = g[prev][v[j]] += g[v[zj]][v[j]];
                v[zj] = v[--n];
                break;
            }
            prev = v[zj];

            // update the weights of its neighbours
            for( int j = 1; j < n; j++ ) if( !a[v[j]] )
                w[j] += g[v[zj]][v[j]];
        }
    }
    return best;
}
```

4.6 Directed minimum spanning tree $O(m \log n)$

```
/*
  GETS:
  call make_graph(n) at first
  you should use add_edge(u,v,w) and
  add pair of vertices as edges (vertices are 0..n-1)

  GIVES:
  output of dmst(v) is the minimum arborescence with root v in
      directed graph
  (INF if it hasn't a spanning arborescence with root v)

  O(mlogn)
*/

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

const int INF = 2e7;

struct MinimumAborescense
{
  struct edge {
    int src, dst, weight;
  };

  struct union_find {
    vector<int> p;
    union_find(int n) : p(n, -1) { };
    bool unite(int u, int v) {
      if ((u = root(u)) == (v = root(v))) return false;
      if (p[u] > p[v]) swap(u, v);
      p[u] += p[v]; p[v] = u;
      return true;
    }
    bool find(int u, int v) { return root(u) == root(v); }
    int root(int u) { return p[u] < 0 ? u : p[u] = root(p[u]); }
    int size(int u) { return -p[root(u)]; }
  };

  struct skew_heap {
    struct node {
```

```
    node *ch[2];
    edge key;
    int delta;
  } *root;
  skew_heap() : root(0) { }
  void propagate(node *a) {
    a->key.weight += a->delta;
    if (a->ch[0]) a->ch[0]->delta += a->delta;
    if (a->ch[1]) a->ch[1]->delta += a->delta;
    a->delta = 0;
  }
  node *merge(node *a, node *b) {
    if (!a || !b) return a ? a : b;
    propagate(a); propagate(b);
    if (a->key.weight > b->key.weight) swap(a, b);
    a->ch[1] = merge(b, a->ch[1]);
    swap(a->ch[0], a->ch[1]);
    return a;
  }
  void push(edge key) {
    node *n = new node();
    n->ch[0] = n->ch[1] = 0;
    n->key = key; n->delta = 0;
    root = merge(root, n);
  }
  void pop() {
    propagate(root);
    node *temp = root;
    root = merge(root->ch[0], root->ch[1]);
  }
  edge top() {
    propagate(root);
    return root->key;
  }
  bool empty() {
    return !root;
  }
  void add(int delta) {
    root->delta += delta;
  }
  void merge(skew_heap x) {
    root = merge(root, x.root);
  }
};

vector<edge> edges;
void add_edge(int src, int dst, int weight) {
  edges.push_back({src, dst, weight});
}
int n;
void make_graph(int _n) {
  n = _n;
  edges.clear();
}

int dmst(int r) {
  union_find uf(n);
  vector<skew_heap> heap(n);
  for (auto e: edges)
    heap[e.dst].push(e);
  double score = 0;
  vector<int> seen(n, -1);
  seen[r] = r;
  for (int s = 0; s < n; ++s) {
    vector<int> path;
    for (int u = s; seen[u] < 0;) {
      path.push_back(u);
      seen[u] = s;
      if (heap[u].empty()) return INF;

      edge min_e = heap[u].top();
      score += min_e.weight;
      heap[u].add(-min_e.weight);
      heap[u].pop();

      int v = uf.root(min_e.src);
      if (seen[v] == s) {
        skew_heap new_heap;
        while (1) {
          int w = path.back();
          path.pop_back();
          new_heap.merge(heap[w]);
          if (!uf.unite(v, w)) break;
        }
        heap[uf.root(v)] = new_heap;
        seen[uf.root(v)] = -1;
      }
      u = uf.root(v);
    }
  }
  return score;
}
};
```

4.7 Directed minimum spanning tree $O(nm)$

```
/*
  GETS:
  call make_graph(n) at first
  you should use add_edge(u,v,w) and
  add pair of vertices as edges (vertices are 0..n-1)

  GIVES:
  output of dmst(v) is the minimum arborescence with root v in
      directed graph
  (-1 if it hasn't a spanning arborescence with root v)

  O(mn)
*/

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

const int INF = 2e7;

struct MinimumAborescense
{
  int n;
  struct edge {
    int src, dst;
    int weight;
  };
  vector<edge> edges;

  void make_graph(int _n) {
    n=_n;
    edges.clear();
  }

  void add_edge(int u, int v, int w) {
    edges.push_back({u, v, w});
  }

  int dmst(int r) {
    int N = n;
    for (int res = 0; ;) {
      vector<edge> in(N, {-1,-1,(int)INF});
      vector<int> C(N, -1);
      for (auto e: edges)
        if (in[e.dst].weight > e.weight)
          in[e.dst] = e;
      in[r] = {r, r, 0};

      for (int u = 0; u < N; ++u) { // no comming edge ==> no
          aborescense
        if (in[u].src < 0) return -1;
        res += in[u].weight;
      }
      vector<int> mark(N, -1); // contract cycles
      int index = 0;
      for (int i = 0; i < N; ++i) {
        if (mark[i] != -1) continue;
        int u = i;
        while (mark[u] == -1) {
          mark[u] = i;
          u = in[u].src;
        }
        if (mark[u] != i || u == r) continue;
        for (int v = in[u].src; u != v; v = in[v].src) C[v] = index;
        C[u] = index++;
      }
      if (index == 0) return res; // found arborescence
      for (int i = 0; i < N; ++i) // contract
        if (C[i] == -1) C[i] = index++;

      vector<edge> next;
      for (auto &e: edges)
        if (C[e.src] != C[e.dst] && C[e.dst] != C[r])
          next.push_back({C[e.src], C[e.dst], e.weight - in[e.dst].
              weight});
      edges.swap(next);
      N = index; r = C[r];
    }
  }
};
```

4.8 Dominator tree

```
struct DominatorTree
{
vector<int> adj[MAXN], radj[MAXN], tree[MAXN], bucket[MAXN];
    // SET MAXIMUM NUMBER OF NODES
int sdom[MAXN], par[MAXN], idom[MAXN], dsu[MAXN], label[MAXN];
int arr[MAXN], rev[MAXN], cnt;
void clear()
{
        for (int i = 0; i < MAXN; i++)
        {
                adj[i].clear();
                radj[i].clear();
                tree[i].clear();
                sdom[i] = idom[i] = dsu[i] = label[i] = i;
                arr[i] = -1;
        }
        cnt = 0;
}
void add_edge(int u, int v)
{
        adj[u].push_back(v);
}
void dfs(int v)
{
        arr[v] = cnt;
        rev[cnt] = v;
        cnt++;
        for (int i = 0; i < adj[v].size(); i++)
        {
                int u = adj[v][i];
                if (arr[u] == -1)
                {
                        dfs(u);
                        par[arr[u]] = arr[v];
                }
                radj[arr[u]].push_back(arr[v]);
        }
}
int find(int v, int x = 0)
{
        if (dsu[v] == v)
                return (x ? -1 : v);
        int u = find(dsu[v], x + 1);
        if (u < 0)
                return v;
        if (sdom[label[dsu[v]]] < sdom[label[v]])
                label[v] = label[dsu[v]];
        dsu[v] = u;
        return (x ? u : label[v]);
}
void merge(int u, int v)
{
        dsu[v] = u;
}
void build(int root)
{
        dfs(root);
        int n = cnt;
        for (int v = n - 1; v >= 0; v--)
        {
                for (int i = 0; i < radj[v].size(); i++)
                {
                        int u = radj[v][i];
				sdom[v] = min(sdom[v], sdom[find(u)]);
			}
			if (v > 0)
				bucket[sdom[v]].push_back(v);
			for (int i = 0; i < bucket[v].size(); i++)
			{
				int u = bucket[v][i];
				int w = find(u);
				if (sdom[u] == sdom[w])
					idom[u] = sdom[u];
				else
					idom[u] = w;
			}
			if (v > 0)
				merge(par[v], v);
		}
		for (int v = 1; v < n; v++)
		{
			if (idom[v] != sdom[v])
				idom[v] = idom[idom[v]];
			tree[rev[v]].push_back(rev[idom[v]]);
			tree[rev[idom[v]]].push_back(rev[v]);
		}
	}
	DominatorTree()
	{
		clear();
	}
};
```

5 Combinatorics

5.1 LP simplex

```
// Two-phase simplex algorithm for solving linear programs of the form
//
//     maximize     c^T x
//     subject to   Ax <= b
//                  x >= 0
//
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will be stored
//
// OUTPUT: value of the optimal solution (infinity if unbounded
//         above, nan if infeasible)
//
// To use this code, create an LPSolver object with A, b, and c as
// arguments.  Then, call Solve(x).

#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>
#include <limits>

using namespace std;

typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

const DOUBLE EPS = 1e-9;

struct LPSolver {
  int m, n;
  VI B, N;
  VVD D;

  LPSolver(const VVD &A, const VD &b, const VD &c) :
    m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2, VD(n + 2)) {
    for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] =
        A[i][j];
    for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n] = -1; D[i][n +
         1] = b[i]; }
    for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
    N[n] = -1; D[m + 1][n] = 1;
  }

  void Pivot(int r, int s) {
    double inv = 1.0 / D[r][s];
    for (int i = 0; i < m + 2; i++) if (i != r)
      for (int j = 0; j < n + 2; j++) if (j != s)
        D[i][j] -= D[r][j] * D[i][s] * inv;
    for (int j = 0; j < n + 2; j++) if (j != s) D[r][j] *= inv;
    for (int i = 0; i < m + 2; i++) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }

  bool Simplex(int phase) {
    int x = phase == 1 ? m + 1 : m;
    while (true) {
      int s = -1;
      for (int j = 0; j <= n; j++) {
        if (phase == 2 && N[j] == -1) continue;
        if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j]
             < N[s]) s = j;
      }
      if (D[x][s] > -EPS) return true;
      int r = -1;
      for (int i = 0; i < m; i++) {
        if (D[i][s] < EPS) continue;
        if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s]
            ||
          (D[i][n + 1] / D[i][s]) == (D[r][n + 1] / D[r][s]) && B[i] <
               B[r]) r = i;
      }
      if (r == -1) return false;
      Pivot(r, s);
    }
  }

  DOUBLE Solve(VD &x) {
    int r = 0;
    for (int i = 1; i < m; i++) if (D[i][n + 1] < D[r][n + 1]) r = i;
    if (D[r][n + 1] < -EPS) {
      Pivot(r, n);
      if (!Simplex(1) || D[m + 1][n + 1] < -EPS) return -
          numeric_limits<DOUBLE>::infinity();
      for (int i = 0; i < m; i++) if (B[i] == -1) {
        int s = -1;
        for (int j = 0; j <= n; j++)
          if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[
              j] < N[s]) s = j;
        Pivot(i, s);
      }
    }
    if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
    x = VD(n);
    for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n + 1];
    return D[m][n + 1];
  }
};

int main() {

  const int m = 4;
  const int n = 3;
  DOUBLE _A[m][n] = {
    { 6, -1, 0 },
    { -1, -5, 0 },
    { 1, 5, 1 },
    { -1, -5, -1 }
  };
  DOUBLE _b[m] = { 10, -4, 5, -5 };
  DOUBLE _c[n] = { 1, -1, 0 };

  VVD A(m);
  VD b(_b, _b + m);
  VD c(_c, _c + n);
  for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);

  LPSolver solver(A, b, c);
  VD x;
  DOUBLE value = solver.Solve(x);

  cerr << "VALUE: " << value << endl; // VALUE: 1.29032
  cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
  for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
  cerr << endl;
  return 0;
}
```

5.2 FFT

```
const int LG = 20; // IF YOU WANT TO CONVOLVE TWO ARRAYS OF LENGTH N
    AND M CHOOSE LG IN SUCH A WAY THAT 2^LG > n + m
const int MAX = 1 << LG;

struct point
{
        double real, imag;
        point(double _real = 0.0, double _imag = 0.0)
        {
                real = _real;
                imag = _imag;
        }
};
point operator + (point a, point b)
{
        return point(a.real + b.real, a.imag + b.imag);
}
point operator - (point a, point b)
{
        return point(a.real - b.real, a.imag - b.imag);
}
point operator * (point a, point b)
{
        return point(a.real * b.real - a.imag * b.imag, a.real * b.
            imag + a.imag * b.real);
}

void fft(point *a, bool inv)
{
        for (int mask = 0; mask < MAX; mask++)
        {
                int rev = 0;
                for (int i = 0; i < LG; i++)
                        if ((1 << i) & mask)
                                rev |= (1 << (LG - 1 - i));
                if (mask < rev)
                        swap(a[mask], a[rev]);
        }
        for (int len = 2; len <= MAX; len *= 2)
        {
                double ang = 2.0 * M_PI / len;
                if (inv)
                        ang *= -1.0;
                point wn(cos(ang), sin(ang));
                for (int i = 0; i < MAX; i += len)
                {
                        point w(1.0, 0.0);
                        for (int j = 0; j < len / 2; j++)
                        {
                                point t1 = a[i + j] + w * a[i + j +
                                    len / 2];
                                point t2 = a[i + j] - w * a[i + j +
                                    len / 2];
                                a[i + j] = t1;
                                a[i + j + len / 2] = t2;
                                w = w * wn;
                        }
                }
        }
        if (inv)
                for (int i = 0; i < MAX; i++)
                {
                        a[i].real /= MAX;
                        a[i].imag /= MAX;
                }
}
```

5.3 NTT

```
const int MOD = 998244353;
const int LG = 16; // IF YOU WANT TO CONVOLVE TWO ARRAYS OF LENGTH N
    AND M CHOOSE LG IN SUCH A WAY THAT 2^LG > n + m
const int MAX = (1 << LG);
const int ROOT = 44759; // ENSURE THAT ROOT^2^(LG - 1) = MOD - 1
int bpow(int a, int b)
{
        int ans = 1;
        while (b)
        {
                if (b & 1)
                        ans = 1LL * ans * a % MOD;
                b >>= 1;
                a = 1LL * a * a % MOD;
        }
        return ans;
}
void ntt(int *a, bool inv)
{
        for (int mask = 0; mask < MAX; mask++)
        {
                int rev = 0;
                for (int i = 0; i < LG; i++)
                        if ((1 << i) & mask)
                                rev |= (1 << (LG - 1 - i));
                if (mask < rev)
                        swap(a[mask], a[rev]);
        }
        for (int len = 2; len <= MAX; len *= 2)
        {
                int wn = bpow(ROOT, MAX / len);
                if (inv)
                        wn = bpow(wn, MOD - 2);
                for (int i = 0; i < MAX; i += len)
                {
                        int w = 1;
                        for (int j = 0; j < len / 2; j++)
                        {
                                int l = a[i + j];
                                int r = 1LL * w * a[i + j + len / 2] %
                                     MOD;
                                a[i + j] = (l + r);
                                a[i + j + len / 2] = l - r + MOD;
                                if (a[i + j] >= MOD)
                                        a[i + j] -= MOD;
                                if (a[i + j + len / 2] >= MOD)
                                        a[i + j + len / 2] -= MOD;
                                w = 1LL * w * wn % MOD;
                        }
                }
        }
        if (inv)
        {
                int x = bpow(MAX, MOD - 2);
                for (int i = 0; i < MAX; i++)
                        a[i] = 1LL * a[i] * x % MOD;
        }
}
```

5.4 Base Vectors in Z2

```
struct Base
{
    ll a[B] = {};
    ll eliminate(ll x)
    {
        for(int i=B-1; i>=0; --i) if(x >> i & 1) x ^= a[i];
        return x;
    }
    void add(ll x)
    {
        x = eliminate(x);
        for(int i=B-1; i>=0; --i) if(x >> i & 1)
        {
            a[i] = x;
                        for(int j = i - 1; j >= 0; j--) if(a[j] >> i &
                             1) a[j] ^= x;
            return;
        }
    }
    int size()
    {
        int cnt = 0;
        for(int i=0; i<B; ++i) if(a[i]) ++cnt;
        return cnt;
    }
};
```

6 !]gauss.cpp Gaussian Eliminatio

6.1 Stirling 1

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define pb push_back
const int mod = 998244353;
const int root = 15311432;
const int root_1 = 469870224;
const int root_pw = 1 << 23;
const int N = 400004;

vector<int> v[N];

ll modInv(ll a, ll mod = mod){
        ll x0 = 0, x1 = 1, r0 = mod, r1 = a;
        while(r1){
                ll q = r0 / r1;
                x0 -= q * x1; swap(x0, x1);
                r0 -= q * r1; swap(r0, r1);
        }
        return x0 < 0 ? x0 + mod : x0;
}

void fft (vector<int> &a, bool inv) {
        int n = (int) a.size();

        for (int i=1, j=0; i<n; ++i) {
                int bit = n >> 1;
                for (; j>=bit; bit>>=1)
                        j -= bit;
                j += bit;
                if (i < j)
                        swap (a[i], a[j]);
        }

        for (int len=2; len<=n; len<<=1) {
                int wlen = inv ? root_1 : root;
                for (int i=len; i<root_pw; i<<=1)
                        wlen = int (wlen * 1ll * wlen % mod);
                for (int i=0; i<n; i+=len) {
                        int w = 1;
                        for (int j=0; j<len/2; ++j) {
                                int u = a[i+j],  v = int (a[i+j+len/2]
                                      * 1ll * w % mod);
                                a[i+j] = u+v < mod ? u+v : u+v-mod;
                                a[i+j+len/2] = u-v >= 0 ? u-v : u-v+
                                    mod;
                                w = int (w * 1ll * wlen % mod);
                        }
                }
        }

        if(inv) {
                int nrev = modInv(n, mod);
                for (int i=0; i<n; ++i)
                        a[i] = int (a[i] * 1ll * nrev % mod);
        }
}

void pro(const vector<int> &a, const vector<int> &b, vector<int> &res)
     {
        vector<int> fa(a.begin(), a.end()),  fb(b.begin(), b.end());
        int n = 1;
        while (n < (int) max(a.size(), b.size()))  n <<= 1;
        n <<= 1;
        fa.resize (n),  fb.resize (n);

        fft(fa, false), fft (fb, false);
        for (int i = 0; i < n; ++i)
                fa[i] = 1LL * fa[i] * fb[i] % mod;
        fft (fa, true);
        res = fa;
}

int S(int n, int r) {

        int nn = 1;
        while(nn < n) nn <<= 1;

        for(int i = 0; i < n; ++i) {
                v[i].push_back(i);
                v[i].push_back(1);
        }
        for(int i = n; i < nn; ++i) {
                v[i].push_back(1);
        }

        for(int j = nn; j > 1; j >>= 1){
                int hn = j >> 1;
                for(int i = 0; i < hn; ++i){
                        pro(v[i], v[i + hn], v[i]);
                }
        }

        return v[0][r];
}

int fac[N], ifac[N], inv[N];

void prencr(){
        fac[0] = ifac[0] = inv[1] = 1;
        for(int i = 2; i < N; ++i)
                inv[i] = mod - 1LL * (mod / i) * inv[mod % i] % mod;
        for(int i = 1; i < N; ++i){
                fac[i] = 1LL * i * fac[i - 1] % mod;
                ifac[i] = 1LL * inv[i] * ifac[i - 1] % mod;
        }
}

int C(int n, int r){
        return (r >= 0 && n >= r) ? (1LL * fac[n] * ifac[n - r] % mod
            * ifac[r] % mod) : 0;
}

int main(){
        prencr();
        int n, p, q;
        cin >> n >> p >> q;
        ll ans = C(p + q - 2, p - 1);
        ans *= S(n - 1, p + q - 2);
        ans %= mod;
        cout << ans;
}
```

6.2 Stirling 2

$$\left\{ \begin{array}{c} \text{n} \\ \text{k} \end{array} \right\} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

6.3 Chinese remainder

```
long long GCD(long long a, long long b) { return (b == 0) ? a : GCD(b,
     a % b); }
inline long long LCM(long long a, long long b) { return a / GCD(a, b)
    * b; }
inline long long normalize(long long x, long long mod) { x %= mod; if
    (x < 0) x += mod; return x; }
struct GCD_type { long long x, y, d; };
GCD_type ex_GCD(long long a, long long b)
{
    if (b == 0) return {1, 0, a};
    GCD_type pom = ex_GCD(b, a % b);
    return {pom.y, pom.x - a / b * pom.y, pom.d};
}
int testCases;
int t;
long long r[N], n[N], ans, lcm;
int main()
{
    cin >> t;
    for(int i = 1; i <= t; i++) cin >> r[i] >> n[i], normalize(r[i], n
        [i]);
    ans = r[1];
    lcm = n[1];
    for(int i = 2; i <= t; i++)
    {
        auto pom = ex_GCD(lcm, n[i]);
        int x1 = pom.x;
        int d = pom.d;
        if((r[i] - ans) % d != 0) return cerr << "No solutions" <<
            endl, 0;
        ans = normalize(ans + x1 * (r[i] - ans) / d % (n[i] / d) * lcm
            , lcm * n[i] / d);
        lcm = LCM(lcm, n[i]); // you can save time by replacing above
            lcm * n[i] /d by lcm = lcm * n[i] / d
    }
    cout << ans << " " << lcm << endl;

    return 0;
}
```

6.4 Popular LP

BellmanFord:
maximize X_n
$X_1 = 0$
and for eache edge ($v- > u$ and weight w):
$X_u - X_v \leq w$

Flow:
maximize Σf_{out} (where *out* is output edges of vertex 1)
for each vertex (except 1 and n):
$\Sigma f_{in} - \Sigma f_{out} = 0$ (where *in* is input edges of v and *out* is output edges of v)

Dijkstra(IP):
minimize $\Sigma z_i * w_i$
for eache edge ($v- > u$ and weight w):
$0 \leq z_i \leq 1$
and for each ST-cut which vertex 1 is in S and vertex n is in T:
$\Sigma z_e \geq 1$ (for each edge e from S to T)

6.5 Duality of LP

primal: Maximize $c^T x$ subject to $Ax \leq b, x \geq 0$
dual: Minimize $b^T y$ subject to $A^T y \geq c, y \geq 0$

6.6 Extended catalan

number of ways for going from 0 to A with k moves without going to -B:

$$\binom{k}{\frac{A+k}{2}} - \binom{k}{\frac{2B+A+k}{2}}$$

6.7 Find polynomial from it's points

$P(x) = \sum_{i=1}^{n} y_i \prod_{j=1, j\neq i}^{n} \frac{x-x_j}{x_i-x_j}$

7 Constants

7.1 Number of primes

```
30: 10
60: 17
100: 25
1000: 168
10000: 1229
100000: 9592
1000000: 78498
10000000: 664579
```

7.2 Factorials

```
1: 1
2: 2
3: 6
4: 24
5: 120
6: 720
7: 5040
8: 40320
9: 362880
10: 3628800
11: 39916800
12: 479001600
13: 6227020800
14: 87178291200
15: 1307674368000
```

7.3 Powers of 3

```
1: 3
2: 9
3: 27
4: 81
5: 243
```

```
6: 729
7: 2187
8: 6561
9: 19683
10: 59049
11: 177147
12: 531441
13: 1594323
14: 4782969
15: 14348907
16: 43046721
17: 129140163
18: 387420489
19: 1162261467
20: 3486784401
```

7.4 C(2n,n)

```
1: 2
2: 6
3: 20
4: 70
5: 252
6: 924
7: 3432
8: 12870
9: 48620
10: 184756
11: 705432
12: 2704156
13: 10400600
14: 40116600
15: 155117520
```

7.5 Most divisor

```
<= 1e2: 60 with 12 divisors
<= 1e3: 840 with 32 divisors
<= 1e4: 7560 with 64 divisors
<= 1e5: 83160 with 128 divisors
<= 1e6: 720720 with 240 divisors
<= 1e7: 8648640 with 448 divisors
<= 1e8: 73513440 with 768 divisors
<= 1e9: 735134400 with 1344 divisors
<= 1e10: 6983776800 with 2304 divisors
<= 1e11: 97772875200 with 4032 divisors
<= 1e12: 963761198400 with 6720 divisors
<= 1e13: 9316358251200 with 10752 divisors
<= 1e14: 97821761637600 with 17280 divisors
<= 1e15: 866421317361600 with 26880 divisors
<= 1e16: 8086598962041600 with 41472 divisors
<= 1e17: 74801040398884800 with 64512 divisors
<= 1e18: 897612484786617600 with 103680 divisors
```