15-Puzzle (Manhattan distance + Linear conflict heuristic).c

```c
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>

#define inf 1010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define swap(x, y) (x ^= y, y ^= x, x ^= y)
#define F(i, j) (abs(((ar[i][j] - 1) >> 2) - i) + abs( ((ar[i][j] - 1) &
3) - j))

bool flag;
char str[60];
char dir[] = "LRUD";
int dx[] = {0, 0, -1, 1};
int dy[] = {-1, 1, 0, 0};
int len[4] = {0}, idx[4][4], ar[4][4], transpose[4][4];

int row_conflict(int rc){
   int i, j, k, x, y, res = 0;

   for (i = 0; i < 4; i++){
       x = (ar[rc][i] >> 2);
       if (ar[rc][i] != 16) idx[x][len[x]++] = ar[rc][i];
   }

   for (i = 0; i < 4; i++){
       if (len[i] > 1){
           for (j = 0; j < len[i]; j++){
               for (k = j + 1; k < len[i]; k++){
                   if (idx[i][j] > idx[i][k]) res += 2;
               }
           }
       }
       len[i] = 0;
   }
   return res;
}

int column_conflict(int rc){
   int i, j, k, x, y, res = 0;

   for (i = 0; i < 4; i++){
       x = (ar[i][rc] & 3);
       if (ar[i][rc] != 16) idx[x][len[x]++] = ar[i][rc];
   }

   for (i = 0; i < 4; i++){
       if (len[i] > 1){
           for (j = 0; j < len[i]; j++){
               for (k = j + 1; k < len[i]; k++){
```

```
                    if (idx[i][j] > idx[i][k]) res += 2;
                }
            }
        }
        len[i] = 0;
    }
    return res;
}


int heuristic(int bx, int by){
    int i, j, k, l, res, linear_conflict = 0, manhattan_distance = 0;

    for (i = 0; i < 4; i++){
        for (j = 0; j < 4; j++){
            transpose[j][i] = ar[i][j];
            if (ar[i][j] != 16){
                manhattan_distance += F(i, j);
            }
        }
        linear_conflict += row_conflict(i);
        linear_conflict += column_conflict(i);
    }

    res = manhattan_distance + linear_conflict;
    return res;
}

int ida(int bx, int by, int lx, int ly, int g, int lim, int d, int h){
    if (flag) return;

    if (!h){
        if (!flag){
            flag = true;
            str[d] = 0;
            puts(str);
        }
        return g;
    }

    int f = g + h;
    if (f > lim) return f;

    int i, k, l, nh, r, res = inf;
    for (i = 0; i < 4; i++){
        k = bx + dx[i], l = by + dy[i];
        if (k >= 0 && k < 4 && l >= 0 && l < 4 && !(k == lx && l == ly)){
            nh = h;
            nh -= F(k, l);
            if (bx != k) nh -= row_conflict(bx), nh -= row_conflict(k);
            if (by != l) nh -= column_conflict(by), nh -=
column_conflict(l);
            swap(ar[bx][by], ar[k][l]);
```

```
            nh += F(bx, by);
            if (bx != k) nh += row_conflict(bx), nh += row_conflict(k);
            if (by != l) nh += column_conflict(by), nh +=
column_conflict(l);

            str[d] = dir[i];
            r = ida(k, l, bx, by, g + 1, lim, d + 1, nh);
            swap(ar[bx][by], ar[k][l]);
            if (r < res) res = r;
            if (r <= lim) return r;
        }
    }

    return res;
}

int Solve(int bx, int by){
    int res, lim = heuristic(bx, by);

    flag = false;
    for (; ;){
        res = ida(bx, by, bx, by, 0, lim, 0, heuristic(bx, by));
        if (res <= lim) return res;
        else lim = res;
    }
}

bool Solvable(int bx, int by){
    int i, j, r = 0, counter = 0;

    for (i = 0; i < 16; i++){
        if (ar[i >> 2][i & 3] == 16) r = (i >> 2);
        else{
            for (j = i + 1; j < 16; j++){
                if (ar[j >> 2][j & 3] < ar[i >> 2][i & 3]) counter++;
            }
        }
    }

    return ((counter + r) & 1);
}

int main(){
    int t, i, j, k, bx, by;

    scanf("%d", &t);
    while (t--){
        for (i = 0; i < 4; i++){
            for (j = 0; j < 4; j++){
                scanf("%d", &ar[i][j]);
                if (!ar[i][j]){
                    bx = i, by = j;
                    ar[i][j] = 16;
                }
```

```
            }
        }

        if (!Solvable(bx, by)) puts("This puzzle is not solvable.");
        else{
            int res = Solve(bx, by);
            if (res > 50) puts("This puzzle is not solvable.");
        }
    }
    return 0;
}

/*

5

2 3 4 0
1 5 7 8
9 6 10 12
13 14 11 15


6 2 8 4
12 14 1 10
13 15 3 9
11 0 5 7

6 8 4 2
12 14 1 10
13 15 3 9
11 0 5 7

13 1 2 4
5 0 3 7
9 6 10 12
15 8 11 14

0 12 9 13
15 11 10 14
3 7 2 5
4 8 6 1

*/
```

2D Pattern Matcher.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

const unsigned long long base = 1995433697ULL;
```

```
const unsigned long long fuck = 1000000007ULL;

int n, m, r, c;
char str[2010][2010], pattern[2010][2010];
unsigned long long pattern_hash, P[2010], F[2010], ar[2010];

void Pregenerate(){ /// hash = 271859
   int i, j;

   pattern_hash = 0;
   for (i = 0; i < r; i++){
       unsigned long long res = 0;
       for (j = 0; j < c; j++) res = (res * fuck) + pattern[i][j];
       pattern_hash = (pattern_hash * base) + res;
   }
}

int Solve(){ /// hash = 739899
   int i, j, res = 0;
   unsigned long long x = 0, y;

   for (i = 0; i < r; i++) x = (x * base) + ar[i];
   for (i = 0; i < n; i++){
       if ((i + r) > n) break;
       if (x == pattern_hash) res++;

       y = (x - (P[r - 1] * ar[i]));
       x = (y * base) + ar[i + r];
   }

   return res;
}

int main(){ /// hash = 942531
   int i, j, k, l;

   P[0] = F[0] = 1;
   for (i = 1; i < 2010; i++){
       P[i] = P[i - 1] * base;
       F[i] = F[i - 1] * fuck;
   }

   while (scanf("%d %d %d %d", &r, &c, &n, &m) != EOF){
       for (i = 0; i < r; i++) scanf("%s", pattern[i]);
       for (i = 0; i < n; i++) scanf("%s", str[i]);

       int res = 0;
       Pregenerate();
       for (i = 0; i < n; i++){
           unsigned long long res = 0;
           for (j = 0; j < c; j++) res = (res * fuck) + str[i][j];
           ar[i] = res;
       }
```

```
       for (j = 0; j < m; j++){
           if ((j + c) > m) break;

           res += Solve();
           for (i = 0; i < n; i++){
               unsigned long long x = ar[i];
               x = x - (F[c - 1] * str[i][j]);
               ar[i] = (x * fuck) + str[i][j + c];
           }
       }

       printf("%d\n", res);
   }
   return 0;
}

2D Pattern Matcher.cpp
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 1010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/*** 2D Pattern Matcher, n x m text, r x c pattern ***/

namespace pm{
   const unsigned long long base = 1995433697ULL;
   const unsigned long long base2 = 1000000007ULL;

   int n, m, r, c, match;
   bool occur[MAX][MAX]; /// occur[i][j] = true if pattern occurs in
text[i][j] (pattern[0][0] = text[i][j])
   char str[MAX][MAX], pattern[MAX][MAX];
   unsigned long long pattern_hash, P[MAX], F[MAX], ar[MAX];

   void init(int a, int b, char text[MAX][MAX], int u, int v, char
pat[MAX][MAX]){ /// hash = 502708
       n = a, m = b, r = u, c = v;
       for (int i = 0; i < n; i++) text[i][m] = 0, strcpy(str[i],
text[i]);
       for (int i = 0; i < r; i++) pat[i][c] = 0, strcpy(pattern[i],
pat[i]);

       P[0] = F[0] = 1;
       for (int i = 1; i < MAX; i++){
           P[i] = P[i - 1] * base;
           F[i] = F[i - 1] * base2;
       }
```

```
        pattern_hash = 0;
        for (int i = 0; i < r; i++){
            unsigned long long res = 0;
            for (int j = 0; j < c; j++) res = (res * base2) +
pattern[i][j];
            pattern_hash = (pattern_hash * base) + res;
        }

        for (int i = 0; i < n; i++){
            unsigned long long res = 0;
            for (int j = 0; j < c; j++) res = (res * base2) + str[i][j];
            ar[i] = res;
        }
    }

    void roll(int col){ /// hash = 72374
        int i, j;
        unsigned long long x = 0, y;

        for (i = 0; i < r; i++) x = (x * base) + ar[i];
        for (i = 0; i < n; i++){
            if ((i + r) > n) break;
            if (x == pattern_hash) match++, occur[i][col] = true;

            y = (x - (P[r - 1] * ar[i]));
            x = (y * base) + ar[i + r];
        }
    }

    void solve(){ /// hash = 544808
        match = 0, clr(occur);
        for (int j = 0; j < m; j++){
            if ((j + c) > m) break;

            roll(j);
            for (int i = 0; i < n; i++){
                unsigned long long x = ar[i];
                x = x - (F[c - 1] * str[i][j]);
                ar[i] = (x * base2) + str[i][j + c];
            }
        }
    }
}

int main(){
}
```

2SAT.cpp

```
--------------------------------------------------
#include <stdio.h>
#include <bits/stdtr1c++.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
```

```
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// 2 SAT (1 based index for variables)
/// Each variable can have two possible values (true or false)
/// Variables must satisfy a system of constraints on pairs of variables

namespace sat{
   bool visited[MAX * 2];
   vector <int> adj[MAX * 2], rev[MAX * 2];
   int n, m, l, dfs_t[MAX * 2], order[MAX * 2], parent[MAX * 2];

   inline int neg(int x){
       return ((x) <= n ? (x + n) : (x - n));
   }

   /// Call init once
   void init(int nodes){
       n = nodes, m = nodes * 2;
       for (int i = 0; i < MAX * 2; i++){
           adj[i].clear();
           rev[i].clear();
       }
   }

   /// Add implication, if a then b
   inline void add_implication(int a, int b){
       if (a < 0) a = n - a;
       if (b < 0) b = n - b;

       adj[a].push_back(b);
       rev[b].push_back(a);
   }

   inline void add_or(int a, int b){
       add_implication(-a, b);
       add_implication(-b, a);
   }

   inline void add_xor(int a, int b){
       add_or(a, b);
       add_or(-a, -b);
   }

   inline void add_and(int a, int b){
       add_or(a, b);
       add_or(a, -b);
       add_or(-a, b);
   }
```

```
    /// force variable x to be true (if x is negative, force !x to be
true)
    inline void force_true(int x){
        if (x < 0) x = n - x;
        add_implication(neg(x), x);
    }

    /// force variable x to be false (if x is negative, force !x to be
false)
    inline void force_false(int x){
        if (x < 0) x = n - x;
        add_implication(x, neg(x));
    }

    inline void topsort(int i){
        visited[i] = true;
        int j, x, len = rev[i].size();

        for (j = 0; j < len; j++){
            x = rev[i][j];
            if (!visited[x]) topsort(x);
        }
        dfs_t[i] = ++l;
    }

    inline void dfs(int i, int p){
        parent[i] = p;
        visited[i] = true;
        int j, x, len = adj[i].size();

        for (j = 0; j < len; j++){
            x = adj[i][j];
            if (!visited[x]) dfs(x, p);
        }
    }

    void build(){
        int i, x;
        clr(visited);
        for (i = m, l = 0; i >= 1; i--){
            if (!visited[i]){
                topsort(i);
            }
            order[dfs_t[i]] = i;
        }

        clr(visited);
        for (i = m; i >= 1; i--){
            x = order[i];
            if (!visited[x]){
                dfs(x, x);
            }
        }
    }
```

```
   /// Returns true if the system is 2-satisfiable and returns the
solution (nodes set to true) in vector res
   bool satisfy(vector <int>& res){
       build();
       clr(visited);

       for (int i = 1; i <= m; i++){
           int x = order[i];
           if (parent[x] == parent[neg(x)]) return false;

           if (!visited[parent[x]]){
               visited[parent[x]] = true;
               visited[parent[neg(x)]] = false;
           }
       }

       res.clear();
       for (int i = 1; i <= n; i++){
           if (visited[parent[i]]) res.push_back(i);
       }
       return true;
   }
}

int main(){
   return 0;
}
```

2SAT_OP.cpp

```
#include <bits/stdtr1c++.h>

#define MAX  200010
#define MAXM MAX << 1
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

bool visited[MAX];
int n, m, top, S[MAX], last[MAX];

struct Edge{
   int u, v, next;

   Edge(){
   }

   Edge(int a, int b, int c){
       u = a, v = b, next = c;
   }
};
```

```
int counter = 0;
struct Edge E[MAXM];

void AddEdge(int a, int b){
   if (a < 0) a = n - a;
   if (b < 0) b = n - b;

   E[counter] = Edge(a, b, last[a]);
   last[a] = counter++;
}


bool dfs(int x){
   int neg = x - n;
   if (x <= n) neg = x + n;

   if (visited[neg]) return false;
   if (visited[x]) return true;
   visited[x] = true;

   int j, v;
   S[top++] = x;
   for (j = last[x]; ~j; j = E[j].next){
       v = E[j].v;
       if (!dfs(v)) return false;
   }

   return true;
}

bool TwoSAT(){
   int i;
   for (i = 1; i <= n; i++){
       if (!visited[i] && !visited[i + n]){
           top = 0;
           if (!dfs(i)){
               while (top > 0) visited[S[--top]] = 0;
               if (!dfs(i + n)) return false;
           }
       }
   }
   return true;
}

int main(){
   int T = 0, t, i, a, b;

   scanf("%d", &t);
   while (t--){
       counter = 0;
       clr(visited);
       memset(last, -1, sizeof(last));
```

```
        scanf("%d %d", &n, &m);

        for (i = 0; i < m; i++){
            scanf("%d %d", &a, &b);
            AddEdge(-a, b);
            AddEdge(-b, a);
        }

        if (TwoSAT()) printf("Case %d: Yes\n", ++T);
        else printf("Case %d: No\n", ++T);
    }
    return 0;
}
```

Aho Corasick (Dynamic + Global).cpp

```
#include <stdio.h>
#include <bits/stdtr1c++.h>

#define LOG 19
#define LET 26
#define MAX 300300 /// At least MAX + 300
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

set <int> pool;
long long counter[MAX];
int Q[MAX], edge[256], leaf[MAX], fail[MAX], dp[MAX][LET],
trie[MAX][LET];

struct aho_corasick{
    int root;
    vector <int> nodes;
    vector <string> dictionary;

    inline int node(){
        int id = *(pool.begin());
        pool.erase(pool.begin());

            for (int i = 0; i < LET; i++) trie[id][i] = root;
            nodes.push_back(id);
            fail[id] = root, leaf[id] = 0, counter[id] = 0;
            return id;
    }

    inline int size(){
        return dictionary.size();
    }

    void clear(){
        for (auto i: nodes) pool.insert(i);
```

```
        nodes.clear();
        dictionary.clear();
        root = *(pool.begin());
        node();
        for (int i = 0; i < LET; i++) dp[root][i] = root;
        for (int i = 'a'; i <= 'z'; i++) edge[i] = i - 'a'; /// change
here if different character set
    }

    aho_corasick(){
        clear();
    }

    inline int next(int cur, char ch){
        int x = edge[ch];
        cur = dp[cur][x];
        if (trie[cur][x] != root) cur = trie[cur][x];
        return cur;
    }

    inline void insert(const char* str){
        int j, x, cur = root;
        for (j = 0; str[j] != 0; j++){
            x = edge[str[j]];
            if (trie[cur][x] == root) trie[cur][x] = node();
            cur = trie[cur][x];
        }

        leaf[cur]++;
        dictionary.push_back(str);
    }


    inline void build(){ /// remember to call build
        vector <pair<int, pair<int, char> > > Q;
        Q.push_back({root, {root, 0}});

        for(int i = 0; i < Q.size(); i++){
            int u = Q[i].first;
            int p = Q[i].second.first;
            char c = Q[i].second.second;
            for (int j = 0; j < LET; j++){
                if (trie[u][j] != root) Q.push_back({trie[u][j], {u, j}});
            }

            if (u != root){
                int f = fail[p];
                while (f != root && trie[f][c] == root) f = fail[f];
                if(trie[f][c] != root && trie[f][c] != u) fail[u] =
trie[f][c];
                counter[u] = leaf[u] + counter[fail[u]];

                for (int j = 0; j < LET; j++){
```

```
                    dp[u][j] = u;
                    if (u && trie[u][j] == root) dp[u][j] = 
dp[fail[u]][j];
                }
            }
        }
    }

    inline long long count(const char* str){ /// total number of 
occurrences of all words from dictionary in str
        int cur = root;
        long long res = 0;

        for (int j = 0; str[j]; j++){
            cur = next(cur, str[j]);
            res += counter[cur];
        }
        return res;
    }
};

struct dynamic_aho{ /// dynamic aho corasick in N log N
    aho_corasick ar[LOG];

    dynamic_aho(){
        for (int i = 0; i < LOG; i++) ar[i] = aho_corasick();
    }

    inline void insert(const char* str){
        int i, k = 0;
        for (k = 0; k < LOG && ar[k].size(); k++){}

        ar[k].insert(str);
        for (i = 0; i < k; i++){
            for (auto s: ar[i].dictionary){
                ar[k].insert(s.c_str());
            }
            ar[i].clear();
        }
        ar[k].build();
    }

    inline long long count(const char* str){
        long long res = 0;
        for (int i = 0; i < LOG; i++) res += ar[i].count(str);
        return res;
    }

    inline void clear(){
        for (int i = 0; i < LOG; i++) ar[i].clear();
    }
};

char str[MAX];
```

```
int main(){
   for (int i = 1; i < MAX; i++) pool.insert(pool.end(), i); /// must do
this before anything

   dynamic_aho ar[2];
   int t, i, j, k, l, flag;

   scanf("%d", &t);
   while (t--){
       scanf("%d %s", &flag, str);
       if (flag == 3){
           printf("%lld\n", ar[0].count(str) - ar[1].count(str));
           fflush(stdout);
       }
       else ar[flag - 1].insert(str);
   }
   return 0;
}
```

Aho Corasick (Dynamic + Map).cpp

```
#include <bits/stdtr1c++.h>

#define LOG 19
#define LET 26
#define MAX 300010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

struct aho_corasick{
   int id, edge[256];
   vector <long long> counter;
   vector <string> dictionary;
   vector <map<char, int> > trie;
   vector <int> leaf, fail, dp[LET];

   inline int node(){
       leaf.push_back(0);
       counter.push_back(0);
            trie.push_back(map<char, int>());
            return id++;
   }

   inline int size(){
       return dictionary.size();
   }

   void clear(){
       trie.clear();
```

```
        dictionary.clear();
        fail.clear(), leaf.clear(), counter.clear();
        for (int i = 0; i < LET; i++) dp[i].clear();

        id = 0, node();
        for (int i = 'a'; i <= 'z'; i++) edge[i] = i - 'a'; /// change
here if different character set
    }

    aho_corasick(){
        clear();
    }

    inline void insert(const char* str){
        int j, x, cur = 0;
        for (j = 0; str[j] != 0; j++){
            x = edge[str[j]];
            if (!trie[cur].count(x)){
                int next_node = node();
                trie[cur][x] = next_node;
            }
            cur = trie[cur][x];
        }

        leaf[cur]++;
        dictionary.push_back(str);
    }

    inline void build(){ /// remember to call build
        vector <pair<int, pair<int, int> > > Q;
        fail.resize(id, 0);
        Q.push_back({0, {0, 0}});

        for (int i = 0; i < LET; i++) dp[i].resize(id, 0);
        for (int i = 0; i < id; i++){
            for (int j = 0; j < LET; j++){
                dp[j][i] = i;
            }
        }

        for(int i = 0; i < Q.size(); i++){
            int u = Q[i].first;
            int p = Q[i].second.first;
            char c = Q[i].second.second;
            for(auto& it: trie[u]) Q.push_back({it.second, {u,
it.first}});

            if (u){
                int f = fail[p];
                while (f && !trie[f].count(c)) f = fail[f];
                if(!trie[f].count(c) || trie[f][c] == u) fail[u] = 0;
                else fail[u] = trie[f][c];
                counter[u] = leaf[u] + counter[fail[u]];
```

```
                for (int j = 0; j < LET; j++){
                    if (u && !trie[u].count(j)) dp[j][u] = dp[j][fail[u]];
                }
            }
        }
    }

    inline int next(int cur, char ch){
        int x = edge[ch];
        cur = dp[x][cur];
        if (trie[cur].count(x)) cur = trie[cur][x];
        return cur;
    }

    long long count(const char* str){ /// total number of occurrences of
all words from dictionary in str
        int cur = 0;
        long long res = 0;

        for (int j = 0; str[j] && id > 1; j++){ /// id > 1 because build
will not be called if empty dictionary in dynamic aho corasick
            cur = next(cur, str[j]);
            res += counter[cur];
        }
        return res;
    }
};

struct dynamic_aho{ /// dynamic aho corasick in N log N
    aho_corasick ar[LOG];

    dynamic_aho(){
        for (int i = 0; i < LOG; i++) ar[i].clear();
    }

    inline void insert(const char* str){
        int i, k = 0;
        for (k = 0; k < LOG && ar[k].size(); k++){}

        ar[k].insert(str);
        for (i = 0; i < k; i++){
            for (auto s: ar[i].dictionary){
                ar[k].insert(s.c_str());
            }
            ar[i].clear();
        }
        ar[k].build();
    }

    long long count(const char* str){
        long long res = 0;
        for (int i = 0; i < LOG; i++) res += ar[i].count(str);
        return res;
    }
```

```
};

char str[MAX];

int main(){
   dynamic_aho ar[2];
   int t, i, j, k, l, flag;

   scanf("%d", &t);
   while (t--){
       scanf("%d %s", &flag, str);
       if (flag == 3){
           printf("%lld\n", ar[0].count(str) - ar[1].count(str));
           fflush(stdout);
       }
       else ar[flag - 1].insert(str);
   }
   return 0;
}
```

Aho Corasick (Dynamic).cpp

```
#include <bits/stdtr1c++.h>

#define LOG 19
#define LET 26
#define MAX 300010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

struct aho_corasick{
   long long counter[MAX];
   vector <int> dp[LET];
   vector <string> dictionary;
   vector <vector<int> > trie;
   int id, len, Q[MAX], fail[MAX], edge[256];

   inline int node(){
           trie.push_back(vector<int>(LET, 0));
           fail[id] = 0, counter[id] = 0;
           return id++;
   }

   inline int size(){
       return dictionary.size();
   }

   void clear(){
       trie.clear();
       dictionary.clear();
       id = 0, len = 0, node();
```

```
        for (int i = 0; i < LET; i++) dp[i].clear();
        for (int i = 'a'; i <= 'z'; i++) edge[i] = i - 'a'; /// change
here if different character set
    }

    aho_corasick(){
        clear();
    }

    inline void insert(const char* str){
        int j, x, cur = 0;
        for (j = 0; str[j] != 0; j++){
            x = edge[str[j]];
            if (!trie[cur][x]) trie[cur][x] = node();
            cur = trie[cur][x];
        }

        counter[cur] = 1;
        dictionary.push_back(str);
    }

    inline void build(){ /// remember to call build
        for (int i = 0; i < LET; i++) dp[i].resize(id, 0);
        for (int i = 0; i < id; i++){
            for (int j = 0; j < LET; j++){
                dp[j][i] = i;
            }
        }

        int i, j, x, f = 0, l = 0;
        for (i = 0; i < LET; i++){
            if (trie[0][i]) Q[l++] = trie[0][i];
        }

        while (f < l){
            i = Q[f++];
            for (j = 0; j < LET; j++){
                int &v = trie[i][j];
                if (v == 0) v = trie[fail[i]][j];
                else{
                    Q[l++] = v;
                    fail[v] = trie[fail[i]][j], counter[v] +=
counter[fail[v]];
                }
                if (i && !trie[i][j]) dp[j][i] = dp[j][fail[i]];
            }
        }
    }

    inline int next(int cur, char ch){
        int x = edge[ch];
        cur = dp[x][cur];
        if (trie[cur][x]) cur = trie[cur][x];
        return cur;
```

```
    }

    long long count(const char* str){ /// total number of occurrences of
all words from dictionary in str
        int cur = 0;
        long long res = 0;

        for (int j = 0; str[j] && id > 1; j++){
            cur = next(cur, str[j]);
            res += counter[cur];
        }
        return res;
    }
};

struct dynamic_aho{ /// dynamic aho corasick in N log N
    aho_corasick ar[LOG];

    dynamic_aho(){
        for (int i = 0; i < LOG; i++) ar[i].clear();
    }

    inline void insert(const char* str){
        int i, k = 0;
        for (k = 0; k < LOG && ar[k].size(); k++){}

        ar[k].insert(str);
        for (i = 0; i < k; i++){
            for (auto s: ar[i].dictionary){
                ar[k].insert(s.c_str());
            }
            ar[i].clear();
        }
        ar[k].build();
    }

    long long count(const char* str){
        long long res = 0;
        for (int i = 0; i < LOG; i++) res += ar[i].count(str);
        return res;
    }
};

char str[MAX];

int main(){
    dynamic_aho ar[2];
    int t, i, j, k, l, flag;

    scanf("%d", &t);
    while (t--){
        scanf("%d %s", &flag, str);
        if (flag == 3){
            printf("%lld\n", ar[0].count(str) - ar[1].count(str));
```

```
            fflush(stdout);
        }
        else ar[flag - 1].insert(str);
    }
    return 0;
}
```

Aho Corasick.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define LET 26
#define MAX 405
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int id, Q[MAX], fail[MAX], lethash[256], trie[MAX][LET];

int node(){
    clr(trie[id]);
    fail[id] = 0;
    return id++;
}

void init(){
    for (id = 'a'; id <= 'z'; id++) lethash[id] = id - 'a';
    id = 0, node();
}

void insert(char* str, int i){
    int j, x, cur = 0;
    for (j = 0; str[j] != 0; j++){
        x = lethash[str[j]];
        if (!trie[cur][x]) trie[cur][x] = node();
        cur = trie[cur][x];
    }
}

void build(){
    int i, j, x, f = 0, l = 0;
    for (i = 0; i < LET; i++){
        if (trie[0][i]) Q[l++] = trie[0][i];
    }

    while (f < l){
        for (j = 0, i = Q[f++]; j < LET; j++){
            if (trie[i][j] == 0) trie[i][j] = trie[fail[i]][j];
            else{
                fail[trie[i][j]] = trie[fail[i]][j];
                Q[l++] = trie[i][j];
            }
        }
```

```
   }
}

int main(){
}
```

Aho Corasick.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define LET 26
#define PAT 1010
#define MAX 300010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/// Aho-Corasick Algorithm for string matching
/// Complexity O(N + M) + O(MAX * LET), N = text length, M = sum of
pattern length, MAX = Total nodes in trie

namespace aho{
   int id, len, counter[MAX]; /// counter[i] contains the number of times
node i is visited in the trie after text traversal
   int Q[MAX], T[MAX], fail[MAX], pos[PAT], lethash[256], trie[MAX][LET];
/// trie[i][j] = next node from node i following character j

   inline int node(){
            clr(trie[id]);
            fail[id] = 0, counter[id] = 0;
            return id++;
   }

   inline void init(){
       id = 0, len = 0, node();
       for (int i = 'a'; i <= 'z'; i++) lethash[i] = i - 'a'; /// Change
here if different character set
   }

   inline void insert(char* str, int i){
       int j, x, cur = 0;
       for (j = 0; str[j] != 0; j++){
           x = lethash[str[j]];
           if (!trie[cur][x]) trie[cur][x] = node();
           cur = trie[cur][x];
       }
       pos[i] = cur; /// saving the index of the current pattern
   }

   inline void build(){
       int i, j, x, f = 0, l = 0;
       for (i = 0; i < LET; i++){
```

```
            if (trie[0][i]) Q[l++] = trie[0][i];
        }

        while (f < l){
            i = Q[f++];
            for (j = 0; j < LET; j++){
                int &v = trie[i][j];
                if (v == 0) v = trie[fail[i]][j];
                else{
                    fail[v] = trie[fail[i]][j];
                    Q[l++] = v, T[len++] = v;
                }
            }
        }
    }

    inline void run(const char* str){
        int i, j, x, cur = 0;
        for (j = 0; str[j] != 0; j++){
            x = lethash[str[j]];
            cur = trie[cur][x];
            counter[cur]++;
        }
        for (i = len - 1; i >= 0; i--) counter[fail[T[i]]] +=
counter[T[i]]; /// add to count of fail node also
    }
}

char str[1000010], pattern[PAT];

int main(){
    int T = 0, t, n, i, j, k;

    scanf("%d", &t);
    while (t--){
        aho::init();
        scanf("%d %s", &n, str);

        for (i = 0; i < n; i++){
            scanf("%s", pattern);
            aho::insert(pattern, i);
        }

        aho::build();
        aho::run(str);

        printf("Case %d:\n", ++T);
        for (i = 0; i < n; i++) printf("%d\n", aho::counter[aho::pos[i]]);
/// contains the count of pattern[i] in the text
    }
    return 0;
}
```

Algorithm X + Dancing Links.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAXR 100010
#define MAXC 100010
#define MAXNODE 100010
/// Define MAX limits appropriately, set to large values for safety

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/// Dancing Links data structure to solve exact cover problems
/// There are some constraints as columns and a number of rows
/// Each row satisfies some constraints
/// Objective is to select a subset of rows so that each constraint is
satisfied exactly once
/// Don't forget to initialize first by calling init()

namespace dlx{ /// hash = 641985
   int row[MAXNODE], col[MAXNODE];
   int L[MAXNODE], R[MAXNODE], U[MAXNODE], D[MAXNODE];
   int n, idx, len, selected_rows[MAXR], column_count[MAXC];

   void init(int ncolumn){ /// initialize first with total number of
columns (1 based)
       clr(column_count);
       n = ncolumn, idx = n + 1;
       for (int i = 0; i <= n; i++) U[i] = D[i] = i, L[i] = i - 1, R[i] =
i + 1;
       L[0] = n, R[n] = 0;
   }

   /// r = index of row (1 based)
   /// the vector columns contain the columns which are satisfied with
this row
   inline void addrow(int r, vector <int>& columns){ /// hash = 438353
       int i, c, l = columns.size(), first = idx;

       for (i = 0; i < l; i++){
           c = columns[i];
           L[idx] = idx - 1, R[idx] = idx + 1, D[idx] = c, U[idx] = U[c];
           D[U[c]] = idx, U[c] = idx, row[idx] = r, col[idx] = c;
           column_count[c]++, idx++; /// column_count[c] is the number of
rows which satisfies constraint column c
       }
       R[idx - 1] = first, L[first] = idx - 1;
   }

   /// Removes column c from the structure
   inline void remove(int c){ /// hash = 377852
       L[R[c]] = L[c], R[L[c]] = R[c];
```

```
        for (int i = D[c]; i != c; i = D[i]){
            for (int j = R[i]; j != i; j = R[j]){
                column_count[col[j]]--;
                U[D[j]] = U[j], D[U[j]] = D[j];
            }
        }
    }

    /// Restores the position of column c in the structure
    inline void restore(int c){ /// hash = 804101
        for (int i = U[c]; i != c; i = U[i]){
            for (int j = L[i]; j != i; j = L[j]){
                column_count[col[j]]++;
                U[D[j]] = j, D[U[j]] = j;
            }
        }

        L[R[c]] = c, R[L[c]] = c;
    }

    /// Recursively enumerate to solve exact cover
    bool algorithmX(int depth){ /// hash = 308201
        if(R[0] == 0){
            len = depth;
            return true;
        }

        int i, j, c = R[0];
        for (i = R[0]; i != 0; i = R[i]){ /// Select a column
deterministically
            if(column_count[i] < column_count[c]) c = i; /// if multiple
columns exist, choose the one with minimum count
        }

        remove(c);
        bool flag = false;
        for (i = D[c]; i != c && !flag; i = D[i]){ /// While solution is
not found
            selected_rows[depth] = row[i];
            for (j = R[i]; j != i; j = R[j]) remove(col[j]);
            flag |= algorithmX(depth + 1); /// May be select rows non-
deterministically here with random_shuffle?
            for (j = L[i]; j != i; j = L[j]) restore(col[j]);
        }

        restore(c);
        return flag;
    }

    bool exact_cover(vector<int>& rows){ /// Returns the subset of rows
satisfying exact cover, false otherwise
        rows.clear();
        if(!algorithmX(0)) return false;
```

```
        for(int i = 0; i < len; i++) rows.push_back(selected_rows[i]);
        return true;
    }
}

namespace sudoku{ /// hash = 633326
    inline int encode(int n, int a, int b, int c){ /// Encode information
        return (a * n * n) + (b * n) + c + 1;
    }

    inline void decode(int n, int v, int& a, int& b, int& c){ /// Decode
information
        v--;
        c = v % n, v /= n;
        b = v % n, a = (v / n) % n;
    }

    bool solve(int n, int ar[25][25]){
        int i, j, k, l, c;
        int m = sqrt(n + 0.5); /// m * m sub-grid
        int ncolumn = n * n * 4; /// n * n for cells, n * n for rows, n *
n for columns and n * n for boxes

        dlx::init(ncolumn);
        for (i = 0; i < n; i++){
            for (j = 0; j < n; j++){
                for (k = 0; k < n; k++){
                    if (ar[i][j] == 0 || ar[i][j] == (k + 1)){
                        vector <int> columns;
                        columns.push_back(encode(n, 0, i, j)); /// Each
cell can contain only 1 value
                        columns.push_back(encode(n, 1, i, k)); /// Row[i]
can contain only ar[i][j], if ar[i][j] != 0, otherwise it can contain any
value
                        columns.push_back(encode(n, 2, j, k)); ///
Column[j] can contain only ar[i][j], if ar[i][j] != 0, otherwise it can
contain any value
                        columns.push_back(encode(n, 3, (i / m) * m + j /
m, k)); /// Similarly for box numbered i / m * m + j / m

                        /// Current row selection, place number k in the
cell[i][j] and doing so will cover all columns in the columns vector
                        int cur_row = encode(n, i, j, k);
                        dlx::addrow(cur_row, columns);
                    }
                }
            }
        }

        vector<int> res;
        if (!dlx::exact_cover(res)) return false;

        for (l = 0; l < res.size(); l++){
            decode(n, res[l], i, j, k);
```

```
            ar[i][j] = k + 1;
        }
        return true;
    }
}

int main(){

}
```

All Divisors in Range.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>

#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define optimize(str) __attribute__((optimize(str)))

short len[MAX];
int L[MAX], *divisors[MAX];

void Generate(){
   int i, j, k, c, d, x;

   len[0] = len[1] = L[0] = L[1] = 1;
   for (i = 4; i < MAX; i++, i++) len[i] = 2;

   for (i = 3; (i * i) < MAX; i += 2){
       if (!len[i]){
           for (j = (i * i), d = i << 1; j < MAX; j += d){
               if (!len[j]) len[j] = i;
           }
       }
   }

   for (i = 2; i < MAX; i++){
       if (!len[i]) L[i] = i;
       else{
           L[i] = len[i];
           x = L[i /len[i]];
           if (x > L[i]) L[i] = x;
       }
   }

   divisors[1] = (int *) malloc(sizeof(int));
   divisors[1][0] = 1, len[1] = 1;

   for (i = 2; i < MAX; i++){
       c = 0, k = i;
       while (k > 1 && L[k] == L[i]){
```

```
            c++;
            k /= L[k];
        }

        len[i] = (c + 1) * len[k];
        divisors[i] = (int *) malloc(sizeof(int) * len[i]);

        for (x = 1, j = 0, len[i] = 0; j <= c; j++, x *= L[i]){
            for (d = 0; d < len[k]; d++){
                divisors[i][len[i]++] = divisors[k][d] * x;
            }
        }
    }
}

int main(){
    Generate();
    return 0;
}
```

Alpha Beta Pruning.cpp

```
---------------------------------------------------
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// alpha = maximum score maximizing player (p = 0) is assured so far
/// beta = minimum score minimizing player (p = 1) is assured so far
int backtrack(struct node cur, int i, int p, int alpha, int beta){
    if (i == n){
        return evaluate(cur);
    }

    int j, k, x;
    vector <struct node> ar = transition(cur, deck[i], p);

    for (j = 0; j < ar.size(); j++){
        x = backtrack(ar[j], i + 1, p ^ 1, alpha, beta);
        if (p == 0 && x > alpha) alpha = x; /// First player will maximize
his score
        if (p == 1 && x < beta) beta = x; /// Second player will minimize
his score
        if (alpha >= beta) break;
    }

    if (p == 0) return alpha;
    return beta;
}
```

```
int main(){
   struct node start;
   backtrack(start, 0, 0, -INF, INF);
   return 0;
}
```

Anti Polynomial Hash.c

```
/// Petr Mitrichev Contest 10, Problem F Hash
/// Given a polynomial hash function with base b and mod m, find two bit strings such that their hash is same
/// 2 <= b <= m - 2 and 4 <= m <= 10^14

#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <assert.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

long long b, m, expo[67], dp[4][1 << 16];

long long multiply(long long a, long long b, long long m){
   long long res = 0;
   while (b){
       if (b & 1) res = (res + a) % m;
       a = (a << 1) % m, b >>= 1;
   }
   return res;
}

long long hash(unsigned long long x){
   int i = 0;
   long long h = 0;
   while (x){
       h += dp[i++][x & 65535];
       x >>= 16;
   }
   return (h % m) ^ 0x6663E5667CC492E5LL;
}

void print(long long x){
   int i;
   for (i = 63; i >= 0; i--) printf("%d", (x & (1LL << i)) ? 1 : 0);
   puts("");
}

int main(){
   int i, j, k;
   long long u, v, x, y, s;

   while (scanf("%lld %lld", &b, &m) != EOF){
       if (b == 0 && m == 0) break;
```

```
        for (expo[0] = 1, i = 1; i < 64; i++) expo[i] = multiply(expo[i -
1], b, m);
        for (i = 0; i < 4; i++){
            for (j = 0; j < (1 << 16); j++){
                dp[i][j] = 0;
                for (k = 0; k < 16; k++){
                    dp[i][j] += expo[16 * i + k];
                    if (j & (1 << k)) dp[i][j] += expo[16 * i + k];
                    dp[i][j] %= m;
                }
            }
        }

        x = y = s = 29996224275833LL;
        while (1){
            x = hash(x);
            y = hash(hash(y));
            if (x == y) break;
        }

        while (x != s){
            u = hash(x), v = hash(s);
            if (u == v) break;
            x = u, s = v;
        }

        print(x);
        print(s);
        assert(u == v && x != s);
    }
    return 0;
}
```

Anti Polynomial Hash.cpp

```
#include <stdio.h>
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

namespace hsh{
    long long b[2], m[2], expo[2][67], dp[2][4][1 << 16];

    /// Returns the hash value of character ch in defender's code (set to
mine)
    inline long long char_hash(char ch){
        if (ch == '0') return ch + 1007;
        if (ch == '1') return ch + 1007;
```

```
    }

    /// Merge two double hash values (probably no change required here)
    inline long long merge(long long h1, long long h2){
        return (h1 << 32) ^ h2;
    }

    long long multiply(long long a, long long b, long long m){
        a %= m, b %= m;
        long long res = 0;
        while (b){
            if (b & 1) res = (res + a) % m;
            a = (a << 1) % m, b >>= 1;
        }
        return res;
    }

    long long get_hash(unsigned long long x){
        long long i = 0, h1 = 0, h2 = 0;
        while (x){
            h1 += dp[0][i][x & 65535];
            h2 += dp[1][i][x & 65535];
            i++, x >>= 16;
            if (h1 >= m[0]) h1 -= m[0];
            if (h2 >= m[1]) h2 -= m[1];
        }

        return merge(h1, h2) ^ 0x5555555555555555LL;
    }

    void print(long long x){
        int i;
        for (i = 63; i >= 0; i--) printf("%d", (x & (1LL << i)) ? 1 : 0);
        puts("");
    }

    inline void solve(long long base1, long long mod1, long long base2,
long long mod2){
        int l, i, j, k;
        long long u, v, x, y, s;
        b[0] = base1, m[0] = mod1, b[1] = base2, m[1] = mod2;

        for (l = 0; l < 2; l++){
            for (expo[l][0] = 1, i = 1; i < 64; i++) expo[l][i] =
multiply(expo[l][i - 1], b[l], m[l]);
            for (i = 0; i < 4; i++){
                for (j = 0; j < (1 << 16); j++){
                    dp[l][i][j] = 0;
                    for (k = 0; k < 16; k++){
                        if (j & (1 << k)) dp[l][i][j] +=
multiply(expo[l][16 * i + k], char_hash('1'), m[l]);
                        else dp[l][i][j] += multiply(expo[l][16 * i + k],
char_hash('0'), m[l]);
                        dp[l][i][j] %= m[l];
```

```
                    }
                }
            }
        }

        x = y = s = 6666666666666666677LL;
        while (1){
            x = get_hash(x);
            y = get_hash(get_hash(y));
            if (x == y) break;
        }

        while (x != s){
            u = get_hash(x), v = get_hash(s);
            if (u == v) break;
            x = u, s = v;
        }

        print(x);
        print(s);
        assert(u == v && x != s);
    }
}

int main(){
    long long b1, m1, b2, m2; /// 1000000007 2091573227 1997293877
2117566807

    while (cin >> b1 >> m1 >> b2 >> m2){
        hsh::solve(b1, m1, b2, m2);
    }
    return 0;
}
```

Articulation Point.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/// Note: 0-based indexing for graphs
namespace ap{
    vector <int> adj[MAX];
    bool visited[MAX], cut[MAX];
    int n, disc_t, discover[MAX], low[MAX];

    void init(int nodes){
        n = nodes;
        for (int i = 0; i < MAX; i++) adj[i].clear();
```

```
    }

    void dfs(int i, int p){
        visited[i] = true;
        discover[i] = low[i] = ++disc_t;
        int j, x, children = 0, len = adj[i].size();

        for (j = 0; j < len; j++){
            x = adj[i][j];
            if (!visited[x]){
                children++;
                dfs(x, i);
                low[i] = min(low[i], low[x]);

                if ((low[x] >= discover[i])){
                    if (!(p == -1 && children < 2)){
                        cut[i] = true;
                    }
                }
            }
            else if (x != p) low[i] = min(low[i], discover[x]);
        }
    }

    /// Adds undirected edge from a to b with cost or edge index number i
    /// Note that i is optional, it's not needed to find bridges

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

    void articulate(){
        int i, j;
        clr(visited), clr(cut);

        for (i = 0; i < n; i++){
            if (!visited[i]){
                disc_t = 0;
                dfs(i, -1);
            }
        }
        /// cut[i] = true if i is an articulation point, false otherwise
    }
}

int main(){
}
```

Assembly.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
```

```
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int popcount(int x){
   int counter = 0;
   __asm__ volatile("POPCNT %1, %0;"
       :"=r"(counter)
       :"r"(x)
       :
   );
   return counter;
}

int lzcount(int x){
   int counter = 0;
   __asm__ volatile("LZCNT %1, %0;"
       :"=r"(counter)
       :"r"(x)
       :
   );
   return counter;
}

/// Returns i if x = 2^i and 0 otherwise
int bitscan(unsigned int x){
   __asm__ volatile("bsf %0, %0" : "=r" (x) : "0" (x));
   return x;
}

double fsqrt(double x){ /// Also works for long double
   __asm__ volatile("fsqrt" : "+t" (x));
   return x;
}

int gcd(int a, int b){
   int res;
   __asm__ volatile(
                     "movl %1, %%eax;"
                     "movl %2, %%ebx;"
                     "repeat_%=:\n"
                     "cmpl $0, %%ebx;"
                     "je terminate_%=\n;"
                     "xorl %%edx, %%edx;"
                     "idivl %%ebx;"
                     "movl %%ebx, %%eax;"
                     "movl %%edx, %%ebx;"
                     "jmp repeat_%=\n;"
                     "terminate_%=:\n"
                     "movl %%eax, %0;"

                     : "=g"(res)
                     : "g"(a), "g"(b)
                     : "eax", "ebx", "edx"
```

```
   );
   return res;
}

int main(){
   printf("%d\n", lzcount(13));
   return 0;
}
```

Berlekamp Massey + Matrix Expo.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAXN 666            /// Size of matrix
#define MOD 1000000007      /// MOD must be prime and greater than 2
#define MOD_THRESHOLD 18    /// MOD * MOD * MOD_THRESHOLD < 2^64

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

namespace bbl{ /// IMPORTANT: MOD must be prime and greater than 2
   struct Matrix{ /// hash = 839399 (for Matrix class)
        int n, ar[MAXN][MAXN];

        Matrix(){}
        Matrix(int dimension, int diagonal = 0){
            clr(ar);
            n = dimension;
            for (int i = 0; i < n; i++) ar[i][i] = diagonal;
        }

        Matrix operator* (const Matrix& other) const{
            Matrix res(n);
            int i, j, k, l, d;

            for(i = 0; i < n; i++){
                for(j = 0; j < other.n; j++){
                    unsigned long long x = 0;
                    for(k = 0; k < n; k += MOD_THRESHOLD){
                        for (l = 0; l < MOD_THRESHOLD && (k + l) < n;
l++){
                            x += ((unsigned long long)ar[i][k + l] *
other.ar[k + l][j]);
                        }
                        x %= MOD;
                    }
                    res.ar[i][j] = x;
                }
            }
```

```
            return res;
        }

        Matrix operator^ (long long n) const{
            Matrix x = *this, res = Matrix(this->n, 1);
            while (n){
                if (n & 1) res = res * x;
                n = n >> 1;
                x = x * x;
            }
            return res;
        }
    };

    int mod_inverse(int x){
        int u = 1, v = 0, t = 0, a = x, b = MOD;
        while (b){
            t = a / b;
            a -= (t * b), u -= (t * v);
            swap(a, b), swap(u, v);
        }
        return (u + MOD) % MOD;
    }

    int convolution(const int* A, const int* B, int n){
        int i = 0, j = 0;
        unsigned long long res = 0;
        for (i = 0; (i + MOD_THRESHOLD) <= n; res %= MOD){
            for (j = 0; j < MOD_THRESHOLD; j++, i++) res += (unsigned long
long)A[i] * B[i];
        }

        for (j = 0; i < n; i++) res += (unsigned long long)A[i] * B[i];
        return res % MOD;
    }

    /// Berlekamp Massey algorithm in O(n ^ 2)
    /// Finds the shortest linear recurrence that will generate the
sequence S and returns it in C
    /// S[i] * C[l - 1] + S[i + 1] * C[l - 2] + ... + S[j] * C[0] = 0

    int berlekamp_massey(vector <int> S, vector <int>& C){ /// hash =
347704
        assert(S.size() && (S.size() % 2) == 0);

        int n = S.size();
        C.assign(n + 1, 0);
        vector <int> T, B(n + 1, 0);
        reverse(S.begin(), S.end());

        C[0] = 1, B[0] = 1;
        int i, j, k, d, x, l = 0, m = 1, b = 1, deg = 0;

        for (i = 0; i < n; i++){
```

```
            d = S[n - i - 1];
            if (l > 0) d = (d + convolution(&C[1], &S[n - i], l)) % MOD;
            if (d == 0) m++;
            else{
                if (l * 2 <= i) T.assign(C.begin(), C.begin() + l + 1);
                x = (((long long)mod_inverse(b) * (MOD - d)) % MOD + MOD)
% MOD;

                for (j = 0; j <= deg; j++){
                    C[m + j] = (C[m + j] + (unsigned long long)x * B[j]) %
MOD;
                }
                if (l * 2 <= i){
                    B.swap(T);
                    deg = B.size() - 1;
                    b = d, m = 1, l = i - l + 1;
                }
                else m++;
            }
        }

        C.resize(l + 1);
        return l;
    }

    /// find's the n'th term of the recurrence from a continuous sequence
of recurrence values
    /// recurrence and n both follow 0 based indexing
    long long interpolate(vector <int>& recurrence, long long n){ /// hash
= 377697
        int len = recurrence.size();
        if (n < len) return recurrence[n];

        vector <int> polynomial;
        int l = berlekamp_massey(recurrence, polynomial);
        while (l && polynomial[l] == 0){
            l--;
            polynomial.pop_back();
        }
        reverse(polynomial.begin(), polynomial.begin() + l + 1);

        struct Matrix mat = Matrix(l);
        for (int i = 1; i < mat.n; i++) mat.ar[i][i - 1] = 1;
        for (int i = 0; i < mat.n; i++) mat.ar[0][i] = MOD - polynomial[l
- i - 1];

        long long res = 0;
        mat = mat ^ (n - len + 1);
        for (int i = 0; i < mat.n; i++) res = (res + (long
long)mat.ar[0][i] * recurrence[len - i - 1]) % MOD;
        return res;
    }
}
```

```
int main(){
   vector <int> recurrence = {0, 0, 1, 1, 2, 3}; /// fibonacci sequence
   printf("%lld\n", bbl::interpolate(recurrence, 1234567)); /// output =
481148194
   return 0;
}
```

Berlekamp Massey.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 65536
#define MOD 1000000007 /// MOD must be prime and greater than 2
#define MOD_THRESHOLD 18 /// MOD * MOD * MOD_THRESHOLD < 2^64

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

namespace pol{ /// Polynomial namespace
   unsigned long long temp[128];
   int ptr = 0, A[MAX * 2], B[MAX * 2], buffer[MAX * 6];

   /// Use faster multiplication techniques if required
   void karatsuba(int n, int *a, int *b, int *res){ /// hash = 829512
       int i, j, h;
       if (n < 17){
           for (i = 0; i < (n + n); i++) temp[i] = 0;
           for (i = 0; i < n; i++){
               if (a[i]){
                   for (j = 0; j < n; j++){
                       temp[i + j] += ((long long)a[i] * b[j]);
                   }
               }
           }
           for (i = 0; i < (n + n); i++) res[i] = temp[i] % MOD;
           return;
       }

       h = n >> 1;
       karatsuba(h, a, b, res);
       karatsuba(h, a + h, b + h, res + n);
       int *x = buffer + ptr, *y = buffer + ptr + h, *z = buffer + ptr +
h + h;

       ptr += (h + h + n);
       for (i = 0; i < h; i++){
           x[i] = a[i] + a[i + h], y[i] = b[i] + b[i + h];
           if (x[i] >= MOD) x[i] -= MOD;
           if (y[i] >= MOD) y[i] -= MOD;
       }
```

```
        karatsuba(h, x, y, z);
        for (i = 0; i < n; i++) z[i] -= (res[i] + res[i + n]);
        for (i = 0; i < n; i++){
            res[i + h] = (res[i + h] + z[i]) % MOD;
            if (res[i + h] < 0) res[i + h] += MOD;
        }
        ptr -= (h + h + n);
    }

    int mul(int n, int *a, int m, int *b){ /// hash = 903808
        int i, r, c = (n < m ? n : m), d = (n > m ? n : m), *res = buffer
+ ptr;
        r = 1 << (32 - __builtin_clz(d) - (__builtin_popcount(d) == 1));
        for (i = d; i < r; i++) a[i] = b[i] = 0;
        for (i = c; i < d && n < m; i++) a[i] = 0;
        for (i = c; i < d && m < n; i++) b[i] = 0;

        ptr += (r << 1), karatsuba(r, a, b, res), ptr -= (r << 1);
        for (i = 0; i < (r << 1); i++) a[i] = res[i];
        return (n + m - 1);
    }

    inline void copy(int* to, const int* from, int n){
        memcpy(to, from, n * sizeof(int));
    }

    inline void add(int* res, const int* P, int pn, const int* Q, int qn){
        for (int i = 0; i < qn; i++){
            res[i] = P[i] + Q[i];
            if (res[i] >= MOD) res -= MOD;
        }
        copy(res + qn, P + qn, pn - qn);
    }

    inline void subtract(int* res, const int* P, int pn, const int* Q, int
qn){
        for(int i = 0; i < qn; ++ i){
            res[i] = P[i] - Q[i];
            if (res[i] < 0) res[i] += MOD;
        }
        copy(res + qn, P + qn, pn - qn);
    }

    inline void shift(int* res, const int* P, int n, int k){
        for (int i = 0; i < n; i++) res[i] = ((long long)P[i] << k) % MOD;
    }

    inline void reverse_poly(int* res, const int* P, int n){
        if (&res[0] == &P[0]) reverse(res, res + n);
        else{
            for (int i = 0; i < n; i++) res[n - i - 1] = P[i];
        }
    }
```

```
struct polynomial{
    vector <int> coefficient;

    polynomial(){}
    polynomial(int c0) : coefficient(1, c0){}
    polynomial(int c0, int c1) : coefficient(2) {coefficient[0] = c0,
coefficient[1] = c1;}

    inline void resize(int n){
        coefficient.resize(n);
    }

    inline int* data(){
        return coefficient.empty() ? 0 : &coefficient[0];
    }

    inline const int* data() const{
        return coefficient.empty() ? 0 : &coefficient[0];
    }

    inline int size() const{
        return coefficient.size();
    }

    inline void set(int i, int x){
        if (size() <= i) resize(i + 1);
        coefficient[i] = x;
    }

    inline void normalize(){
        while (size() && coefficient.back() == 0)
coefficient.pop_back();
    }

    static void multiply(polynomial &res, const polynomial& p, const
polynomial& q){ /// hash = 569213
        if(&res == &p || &res == &q){
            polynomial temp;
            multiply(temp, p, q);
            res = temp;
            return;
        }

        res.coefficient.clear();
        if (p.size() != 0 && q.size() != 0){
            int i, j, l, n = 0, m = 0;
            for (i = 0; i < p.size(); i++) A[n++] = p.coefficient[i];
            for (i = 0; i < q.size(); i++) B[m++] = q.coefficient[i];

            l = mul(n, A, m, B);
            for (i = 0; i < l; i++) res.coefficient.push_back(A[i]);
        }
    }
```

```
        polynomial inverse(int n) const{
            polynomial res(n);
            res.resize(n);
            find_inverse(res.data(), n, data(), size());
            return res;
        }

        /// hash = 190805
        static void divide(polynomial &quot, polynomial &rem, const
polynomial &p, const polynomial &q, const polynomial &inv) {
            int pn = p.size(), qn = q.size();
            quot.resize(max(0, pn - qn + 1)), rem.resize(qn - 1);
            divide_remainder_inverse(quot.data(), rem.data(), p.data(),
pn, q.data(), qn, inv.data());
            quot.normalize(), rem.normalize();
        }

        polynomial remainder(const polynomial &q, const polynomial &inv)
const{
            polynomial quot, rem;
            divide(quot, rem, *this, q, inv);
            return rem;
        }

        polynomial power(const polynomial &q, long long k) const{ /// hash
= 556205
            int qn = q.size();
            if(qn == 1) return polynomial();
            if(k == 0) return polynomial(1);
            polynomial inv = q.inverse(max(size() - qn + 1, qn));
            polynomial p = this->remainder(q, inv);

            polynomial res = p;
            int l = 63 - __builtin_clzll(k);

            for(--l; l >= 0; --l){
                multiply(res, res, res);
                res = res.remainder(q, inv);
                if((k >> l) & 1){
                    multiply(res, res, p);
                    res = res.remainder(q, inv);
                }
            }
            return res;
        }

        static void multiply(int* res, const int* P, int pn, const int* Q,
int qn);

        static void inverse_power_series(int* res, int res_n, const int*
P, int pn);
        static void find_inverse(int* res, int res_n, const int* P, int
pn);
```

```
       static void divide_inverse(int* res, int res_n, const int* revp, 
int pn, const int* inv);
       static void divide_remainder_inverse(int* quot, int* rem, const 
int* P, int pn, const int* Q, int qn, const int* inv);
   };

   void polynomial::multiply(int* res, const int* P, int pn, const int* 
Q, int qn){ /// hash = 25722
       polynomial P1, P2, P_res;
       for (int i = 0; i < pn; i++) P1.coefficient.push_back(P[i]);
       for (int i = 0; i < qn; i++) P2.coefficient.push_back(Q[i]);
       P_res.multiply(P_res, P1, P2);
       for (int i = 0; i < P_res.coefficient.size(); i++) res[i] = 
P_res.coefficient[i];
   }

   void polynomial::inverse_power_series(int* res, int res_n, const int* 
P, int pn){ /// hash = 553608
       if(res_n == 0) return;
       unique_ptr <int[]> ptr(new int[res_n * sizeof(int)]);
       int* u = ptr.get(), *v = u + res_n * 2, cur = 1, nxt = 1;

       clr(res);
       res[0] = P[0];
       while (cur < res_n){
           nxt = min(res_n, cur * 2);
           multiply(u, res, cur, res, cur);
           multiply(v, u, min(nxt, cur * 2 - 1), P, min(nxt, pn));
           shift(res, res, cur, 1);
           subtract(res, res, nxt, v, nxt);
           cur = nxt;
       }
   }

   void polynomial::find_inverse(int* res, int res_n, const int* P, int 
pn){
       unique_ptr <int[]> ptr(new int[pn]);
       int* tmp = ptr.get();
       reverse_poly(tmp, P, pn);
       inverse_power_series(res, res_n, tmp, pn);
   }

   void polynomial::divide_inverse(int* res, int res_n, const int* revp, 
int pn, const int* inv){
       unique_ptr <int[]> ptr(new int[pn + res_n]);
       int* tmp = ptr.get();
       multiply(tmp, revp, pn, inv, res_n);
       reverse_poly(res, tmp, res_n);
   }

   /// hash = 203864
   void polynomial::divide_remainder_inverse(int* quot, int* rem, const 
int* P, int pn, const int* Q, int qn, const int* inv){
```

```
        if(pn < qn){
            copy(rem, P, pn);
            for (int i = 0; i < qn - pn - 1; i++) rem[i + pn] = 0;
            return;
        }

        if(qn == 1) return;
        int quot_n = pn - qn + 1;
        int rn = qn - 1, tn = min(quot_n, rn), un = tn + rn;
        unique_ptr <int[]> ptr(new int[pn + un + (quot != 0 ? 0 :
quot_n)]);

        int* revp = ptr.get(), *qmul = revp + pn;
        if(quot == 0) quot = qmul + un;
        reverse_poly(revp, P, pn);
        divide_inverse(quot, quot_n, revp, pn, inv);
        multiply(qmul, Q, rn, quot, tn);
        subtract(rem, P, rn, qmul, rn);
    }
}

namespace bbl{ /// Black box linear algebra
    using namespace pol;

    int mod_inverse(int x){
        int u = 1, v = 0, t = 0, a = x, b = MOD;
        while (b){
            t = a / b;
            a -= (t * b), u -= (t * v);
            swap(a, b), swap(u, v);
        }
        return (u + MOD) % MOD;
    }

    int convolution(const int* A, const int* B, int n){
        int i = 0, j = 0;
        unsigned long long res = 0;
        for (i = 0; (i + MOD_THRESHOLD) <= n; res %= MOD){
            for (j = 0; j < MOD_THRESHOLD; j++, i++) res += (unsigned long
long)A[i] * B[i];
        }

        for (j = 0; i < n; i++) res += (unsigned long long)A[i] * B[i];
        return res % MOD;
    }

    /// Berlekamp Massey algorithm in O(n ^ 2)
    /// Finds the shortest linear recurrence that will generate the
sequence S and returns it in C
    /// S[i] * C[l - 1] + S[i + 1] * C[l - 2] + ... + S[j] * C[0] = 0

    int berlekamp_massey(vector <int> S, vector <int>& C){ /// hash =
768118
        assert((S.size() % 2) == 0);
```

```
        int n = S.size();
        C.assign(n + 1, 0);
        vector <int> T, B(n + 1, 0);
        reverse(S.begin(), S.end());

        C[0] = 1, B[0] = 1;
        int i, j, k, d, x, l = 0, m = 1, b = 1, deg = 0;

        for (i = 0; i < n; i++){
            d = S[n - i - 1];
            if (l > 0) d = (d + convolution(&C[1], &S[n - i], l)) % MOD;
            if (d == 0) m++;
            else{
                if (l * 2 <= i) T.assign(C.begin(), C.begin() + l + 1);
                x = (((long long)mod_inverse(b) * (MOD - d)) % MOD + MOD)
% MOD;

                for (j = 0; j <= deg; j++){
                    C[m + j] = (C[m + j] + (unsigned long long)x * B[j]) %
MOD;
                }
                if (l * 2 <= i){
                    B.swap(T);
                    deg = B.size() - 1;
                    b = d, m = 1, l = i - l + 1;
                }
                else m++;
            }
        }

        C.resize(l + 1);
        return l;
    }

    vector <int> recurrence_coefficients(const vector <int>& recurrence,
long long k){ /// hash = 713914
        polynomial p, res;
        int n = recurrence.size();

        p.resize(n + 1), p.set(n, 1);
        for (int i = 0; i < n; i++) p.set(i, recurrence[i]);

        vector <int> v;
        res = polynomial(0, 1).power(p, k);
        for (int i = 0; i < n && i < res.size(); i++)
v.push_back(res.coefficient[i]);
        return v;
    }

    vector <int> interpolate(const vector<int> &base_sequence, const
vector<int> &polynomial, long long k, int n){ /// hash = 347802
        int i, j, len = polynomial.size() - 1;
        vector <int> recurrence(len);
```

```
        for (i = 0; i < len; i++) recurrence[i] = polynomial[i];
        vector <int> coefficient = recurrence_coefficients(recurrence, k);

        vector <int> res;
        len = min(len, (int)coefficient.size());
        len = min(len, (int)base_sequence.size());
        for (j = 0; j < n; j++){
            long long r = 0;
            for (i = 0; i < len; i++){
                assert((i + j) < base_sequence.size());
                r = (r + (long long)coefficient[i] * base_sequence[i + j])
% MOD;
            }
            res.push_back(r);
        }
        return res;
    }
}

/// Returns consecutive n terms of the recurrence starting from the k'th
term
/// MOD must be prime and greater than 2
/// 0.5s for n <= 4000, 4.25s for n <= 16000

vector <int> interpolate(vector <int>& recurrence, long long k, int n){
    using namespace bbl;
    assert(recurrence.size() && (recurrence.size() % 2) == 0);

    vector <int> polynomial;
    int l = berlekamp_massey(recurrence, polynomial);
       reverse(polynomial.begin(), polynomial.begin() + l + 1);
    return interpolate(recurrence, polynomial, k, n);
}

/// Returns the k'th term of the recurrence

int interpolate(vector <int>& recurrence, long long k){
    vector <int> res = interpolate(recurrence, k, 1);
    return res[0];
}

int main(){
    vector <int> seq = {0, 1, 1, 2, 3, 5, 8, 13};
    long long x, y, m = 1923921584219421LL;
    clock_t start = clock();
    ///for (int i = 0; i < 16000; i++) seq.push_back(rand() + 1);
    x = interpolate(seq, m);
    dbg(x); /// x = 597529871
    fprintf(stderr, "%0.6f\n", (clock() - start) / (1.0 *
CLOCKS_PER_SEC));
    return 0;
}
```

Bernoulli Numbers.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 2510
#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int S[MAX][MAX], inv[MAX], factorial[MAX], bernoulli[MAX];

int expo(long long x, int n){
   long long res = 1;

   while (n){
       if (n & 1) res = (res * x) % MOD;
       x = (x * x) % MOD;
       n >>= 1;
   }

   return (res % MOD);
}

void Generate(){
   int i, j;
   long long x, y, z, lim = (long long)MOD * MOD;

   for (i = 1, factorial[0] = 1; i < MAX; i++) factorial[i] = ((long
long) factorial[i - 1] * i) % MOD;
   for (i = 0; i < MAX; i++) inv[i] = expo(i, MOD - 2);
   for (i = 1, S[0][0] = 1; i < MAX; i++){
       for (j = 1, S[i][0] = 0; j <= i; j++) S[i][j] = ( ((long long)S[i
- 1][j] * j) + S[i - 1][j - 1]) % MOD;
   }

   bernoulli[0] = 1;
   for (i = 1; (i + 1) < MAX; i++){
       if ((i & 1) && i > 1) bernoulli[i] = 0;
       else{
           for (j = 0, x = 0, y = 0; j <= i; j++){
               z = ((long long) factorial[j] * inv[j + 1]) % MOD;
               z = (z * S[i][j]) % MOD;
               if (j & 1) y += z;
               else x += z;
           }
           bernoulli[i] = (lim + x - y) % MOD;
       }
   }
}

int main(){
   Generate();
   printf("%d\n", bernoulli[10]);
```

```
   return 0;
}
```

BigInteger.cpp

```
#include <stdio.h>
#include <bits/stdtr1c++.h>

#define SIZE 100
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

const int base_len = 8;
const long long base = 0x5F5E100;

struct unsigned_bignum{
   int len;
   long long data[SIZE];

   long long &operator [](int x){
       return(data[x]);
   }
      const long long &operator [](int x) const{
          return(data[x]);
   }

   unsigned_bignum(){
       len = 0;
       clr(data);
   }

   void clear(){
       for (int i = 1; i <= len; i++) data[i] = 0;
       len = 0;
   }

   inline int compare(const unsigned_bignum &a, const unsigned_bignum
&b){
       if (a.len > b.len) return 0;
       if (a.len < b.len) return 1;
       for (int i = a.len; i >= 1; i--){
           if (a.data[i] > b.data[i]) return 0;
           if (a.data[i] < b.data[i]) return 1;
       }
       return 2;
   }

   inline bool operator < (const unsigned_bignum &x){
       return (compare(*this, x) == 1);
   }
```

```
   inline bool operator > (const unsigned_bignum &x){
       return (compare(*this, x) == 0);
}
   inline bool operator <= (const unsigned_bignum &x){
       return (compare(*this, x) >= 1);
}
   inline bool operator >= (const unsigned_bignum &x){
       return ((compare(*this, x) & 1) == 0);
}
   inline bool operator != (const unsigned_bignum &x){
       return (compare(*this, x) != 2);
}
   inline bool operator == (const unsigned_bignum &x){
       return (compare(*this, x) == 2);
}
inline unsigned_bignum operator = (const unsigned_bignum& x){
    for (int i = x.len + 1; i <= len; i++) data[i] = 0;
    for (int i = 1; i <= x.len; i++) data[i] = x.data[i];
    len = x.len;
    return *this;
}

inline unsigned_bignum operator = (long long x){
    for (int i = 0; i <= len; i++) data[i] = 0;
    len = 0;

    do{
        data[++len] = x % base;
        x /= base;
    }
    while (x);
    return *this;
}

inline unsigned_bignum operator += (const unsigned_bignum &b){
    return *this = (*this + b);
}
   inline unsigned_bignum operator *= (const unsigned_bignum &b){
       return *this = (*this * b);
}
   inline unsigned_bignum operator -= (const unsigned_bignum &b){
       return *this = (*this - b);
}
   inline unsigned_bignum operator /= (const unsigned_bignum &b){
       return *this = (*this / b);
}
   inline unsigned_bignum operator %= (const unsigned_bignum &b){
       return *this = (*this % b);
}
   inline unsigned_bignum operator *= (long long x){
       return (*this = (*this * x));
}
   inline unsigned_bignum operator += (long long x){
       return (*this = (*this + x));
```

```
}
   inline unsigned_bignum operator -= (long long x){
       return (*this = (*this - x));
}
   inline unsigned_bignum operator /= (long long x){
       return (*this = (*this / x));
}



unsigned_bignum(long long x){
    len = 0;
    clr(data);
    (*this) = x;
}

unsigned_bignum operator * (const unsigned_bignum& x){
    int i, j;
    unsigned_bignum res = unsigned_bignum(0);
    for (i = 1; i <= len; i++){
        if (data[i]){
            for (j = 1; j <= x.len; j++){
                if (x.data[j]){
                    res.data[i + j - 1] += data[i] * x.data[j];
                    res.data[i + j] += res.data[i + j - 1] / base;
                    res.data[i + j - 1] %= base;
                }
            }
        }
    }

    res.len = len + x.len - 1;
        while (res.data[res.len + 1]) res.len++;
        while (res[res.len] == 0 && res.len > 1) res.len--;
        return res;
}

   unsigned_bignum operator / (long long x){
       assert(x != 0);

       int i, j;
       unsigned_bignum res;
       long long y = 0;
       for (i = len; i >= 1; i--){
        y = y * base + data[i];
        res[i] = y / x;
        y %= x;
       }

       res.len = len;
       while (res[res.len] == 0 && res.len > 1) res.len--;
       return res;
   }
```

```
      void divide(const unsigned_bignum& b, unsigned_bignum& rem,
unsigned_bignum& quot){
          assert(!(b.len == 1 && b[1] == 0));

          long long x = data[len], y = b[b.len];
          int i, j, l1 = base_len * (len - 1), l2 = base_len * (b.len -
1);

        while (x) x /= 10, l1++;
        while (y) y /= 10, l2++;

        unsigned_bignum temp = b;
        rem = *this;

        for (i = 1; i * base_len <= (l1 - l2); i++) temp *= base;
        for (i = 1; i <= (l1 - l2) % base_len; i++) temp *= 10;
        for (i = l1 - l2; i >= 0; i--){
            x = 0;
            while (rem >= temp) rem -= temp, x++;
            quot[i / base_len + 1] = quot[i / base_len + 1] * 10 + x;
                  temp /= 10;
        }

        quot.len = (l1 - l2) / base_len + 1;
             while(rem.len > 1 && !rem[rem.len]) rem.len--;
             while(quot.len > 1 && !quot[quot.len]) quot.len--;
      }

      unsigned_bignum operator / (const unsigned_bignum& x){
          unsigned_bignum rem, quot;
          divide(x, rem, quot);
          assert(quot.len > 0);
          return quot;
      }

      unsigned_bignum operator % (const unsigned_bignum& x){
          unsigned_bignum rem, quot;
          divide(x, rem, quot);
          assert(rem.len > 0);
          return rem;
      }

      long long operator % (long long x){
        long long res = 0;
        for (int i = len; i >= 1; i--) res = (res * base + data[i]) % x;
        return res;
   }

      unsigned_bignum operator + (const unsigned_bignum& x){
          unsigned_bignum res;
          int i, l = max(len, x.len);
          for (i = 1; i <= l; i++) res[i] = data[i] + x[i];
          for (i = 1; i <= l; i++) res[i + 1] += res[i] / base, res[i] %=
base;
```

```
        res.len = l;
        if (res[res.len + 1]) res.len++;
        while (res[res.len] == 0 && res.len > 1) res.len--;
        return res;
    }

    unsigned_bignum operator - (const unsigned_bignum& x){
        unsigned_bignum res;
        for (int i = 1; i <= len; i++) res.data[i] = data[i] -
x.data[i];
        for (int i = 1; i <= len; i++){
         if (res[i] < 0) res.data[i] += base, res.data[i + 1]--;
        }

        res.len = len;
        while (res[res.len] == 0 && res.len > 1) res.len--;
        return res;
    }

    unsigned_bignum(const char* str, int dummy){
     int l = strlen(str + 1);
     long long val = 0;
     (*this).clear();

     for (int i = 1; i <= (l - 1) / base_len + 1; i++){
         val = 0;
         for (int j = l - base_len * i + 1; j <= l - base_len * i +
base_len; j++){
             if (j >= 1) val = val * 10 + str[j] - 48;
         }
         data[++len] = val;
     }
    }

  void next_unsigned_bignum(){
      char str[SIZE * base_len + 10];
      scanf("%s", str + 1);
      unsigned_bignum(str, 666);
  }

  void print(char ch = 10){
      printf("%lld", data[len]);
      for (int i = len - 1; i >= 1; i--) printf("%0*lld", base_len,
data[i]);
      putchar(ch);
  }
};

struct bignum{
  int sign;
  unsigned_bignum val;

  bignum(){}
```

```
bignum(long long x){
    sign = 0;
    val = llabs(x);
    if (x < 0) sign = 1;
}

bignum(unsigned_bignum x){
    val = x;
    sign = 0;
}

inline bignum operator = (long long x){
    sign = 0;
    val = llabs(x);
    if (x < 0) sign = 1;
    return *this;
}

bignum operator * (const bignum& x){
    unsigned_bignum y = x.val;
    bignum res;
    res.val = y * this->val;
    res.sign = this->sign ^ x.sign;
    if (res.val == unsigned_bignum(0)) res.sign = 0;
    return res;
}

bignum operator / (const bignum& x){
    unsigned_bignum y = this->val;
    bignum res;
    res.val = y / x.val;
    res.sign = this->sign ^ x.sign;
    if (res.val == unsigned_bignum(0)) res.sign = 0;
    return res;
}

bignum operator % (const bignum& x){
    unsigned_bignum y = this->val;
    bignum res;
    res.val = y % x.val;
    res.sign = this->sign;
    if (res.val == unsigned_bignum(0)) res.sign = 0;
    return res;
}

bignum operator + (const bignum& x){
    bignum res;

    if (this->sign == x.sign){
        res.sign = this->sign;
        res.val = this->val + x.val;
    }
    else{
        unsigned_bignum v = this->val, w = x.val;
```

```
        int cmp = v.compare(v, w);
        if (cmp != 1){
            res.sign = this->sign;
            res.val = v - w;
        }
        else{
            res.sign = x.sign;
            res.val = w - v;
        }
    }

    if (res.val == unsigned_bignum(0)) res.sign = 0;
    return res;
}

bignum operator - (const bignum& x){
    bignum res = x;
    res.sign ^= 1;
    return *this + res;
}


inline int compare(const bignum &a, const bignum &b){
    if (a.sign != b.sign){
        if (a.sign == 0) return 0;
        if (b.sign == 0) return 1;
    }

    unsigned_bignum x = a.val, y = b.val;
    if (a.sign == 0) return x.compare(x, y);
    return x.compare(y, x);
}

inline bool operator < (const bignum &x){
    return (compare(*this, x) == 1);
}
   inline bool operator > (const bignum &x){
       return (compare(*this, x) == 0);
}
   inline bool operator <= (const bignum &x){
       return (compare(*this, x) >= 1);
}
   inline bool operator >= (const bignum &x){
       return ((compare(*this, x) & 1) == 0);
}
   inline bool operator != (const bignum &x){
       return (compare(*this, x) != 2);
}
   inline bool operator == (const bignum &x){
       return (compare(*this, x) == 2);
}

inline bignum operator += (const bignum &b){
```

```
        return *this = (*this + b);
    }
      inline bignum operator *= (const bignum &b){
          return *this = (*this * b);
    }
      inline bignum operator -= (const bignum &b){
          return *this = (*this - b);
    }
      inline bignum operator /= (const bignum &b){
          return *this = (*this / b);
    }
      inline bignum operator %= (const bignum &b){
          return *this = (*this % b);
    }
      inline bignum operator *= (long long x){
          return (*this = (*this * x));
    }
      inline bignum operator += (long long x){
          return (*this = (*this + x));
    }
      inline bignum operator -= (long long x){
          return (*this = (*this - x));
    }
      inline bignum operator /= (long long x){
          return (*this = (*this / x));
      }

      inline void next_bignum(){
          char str[SIZE * base_len + 10];
        scanf("%s", str);
        sign = (str[0] == 45);
        if (!sign){
            int i, l = strlen(str);
            for (i = l; i >= 1; i--) str[i] = str[i - 1];
            str[l + 1] = 0;
        }
        val = unsigned_bignum(str, 666);
      }

    void print(char ch = 10){
        if (sign) putchar(45);
        val.print();
    }
};

int main(){

}
```

Binomial (dp).c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
```

```
#include <stdbool.h>

#define MAX 1010
#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int binomial[MAX][MAX];
long long nPk[MAX][MAX];

void Generate(){
   int i, j;
   clr(binomial);

   for (i = 0; i < MAX; i++){
       for (j = 0; j <= i; j++){
           if (i == j || j == 0) binomial[i][j] = 1;
           else{
               binomial[i][j] = (binomial[i - 1][j] + binomial[i - 1][j -
1]);
               if (binomial[i][j] >= MOD) binomial[i][j] -= MOD;
           }
       }
   }

   clr(nPk);
   for (i = 0; i < MAX; i++){
       for (j = 0; j <= i; j++){
           if (!j) nPk[i][j] = 1;
           else nPk[i][j] = (nPk[i][j - 1] * (i - j + 1)) % MOD;
       }
   }
}

int main(){
   Generate();
}
```

Binomial Coefficients Modulo Prime Powers.cpp

```
#include <stdio.h>
#include <bits/stdtr1c++.h>

#define MAXP 10000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

namespace crt{
   long long extended_gcd(long long a, long long b, long long& x, long
long& y){
```

```
        if (!b){
            y = 0, x = 1;
            return a;
        }

        long long g = extended_gcd(b, a % b, y, x);
        y -= ((a / b) * x);
        return g;
    }

    long long mod_inverse(long long a, long long m){
        long long x, y, inv;
        extended_gcd(a, m, x, y);
        inv = (x + m) % m;
        return inv;
    }

    long long chinese_remainder(vector <long long> ar, vector <long long>
mods){
        int i, j;
        long long x, y, res = 0, M = 1;

        for (i = 0; i < ar.size(); i++) M *= mods[i];
        for (i = 0; i < ar.size(); i++){
            x = M / mods[i];
            y = mod_inverse(x, mods[i]);
            res = (res + (((x * ar[i]) % M) * y)) % M;
        }

        return res;
    }
}

namespace bin{
    int dp[MAXP];
    long long mod = 0;

    long long trailing(long long x, long long p){
        long long res = 0;
        while (x){
            x /= p;
            res += x;
        }
        return res;
    }

    long long expo(long long x, long long n, long long m){
        if (!n) return 1;
        else if (n & 1) return ((expo(x, n - 1, m) * x) % m);
        else{
            long long r = expo(x, n >> 1, m);
            return ((r * r) % m);
        }
    }
```

```
long long factorial(long long x, long long p){
    long long res = expo(dp[mod - 1], x / mod, mod);
    if (x >= p) res = res * factorial(x / p, p) % mod;
    return res * dp[x % mod] % mod;
}

/// binomial co-efficients modulo p^q (p must be a prime)
long long binomial(long long n, long long k, long long p, long long
q){
    if (k > n) return 0;
    if (n == k || k == 0) return 1;

    int i, j;
    for (i = 0, mod = 1; i < q; i++) mod *= p;
    long long t = trailing(n, p) - trailing(k, p) - trailing(n - k,
p);
    if (t >= q) return 0;

    assert(mod < MAXP);
    for (dp[0] = 1, i = 1; i < mod; i++){
        dp[i] = (long long)dp[i - 1] * (i % p ? i : 1) % mod;
    }

    long long res = factorial(n, p) * expo(factorial(k, p) *
factorial(n - k, p) % mod, (mod / p) * (p - 1) - 1, mod) % mod;
    while (t--) res = res * p % mod;
    return res;
}

/// binomial co-efficients modulo m (p_i^q_i of m must be less than
MAXP)
long long binomial(long long n, long long k, long long m){
    if (k > n || m == 1) return 0;
    if (n == k || k == 0) return 1;

    vector <pair<int, int>> factors;
    for (long long i = 2; i * i <= m; i++){
        int c = 0;
        while (m % i == 0){
            c++;
            m /= i;
        }
        if (c) factors.push_back(make_pair(i, c));
    }
    if (m > 1) factors.push_back(make_pair(m, 1));

    vector <long long> ar, mods;
    for (int i = 0; i < factors.size(); i++){
        long long x = 1;
        for (int j = 0; j < factors[i].second; j++) x *=
factors[i].first;
        mods.push_back(x), ar.push_back(binomial(n, k,
factors[i].first, factors[i].second));
```

```
        }
        return crt::chinese_remainder(ar, mods);
    }
}

/// Stress testing below, remove before using
long long dp[101][101];

int main(){
    long long n, k, p, q, m;

    for (p = 2; p <= 10; p++){
        if (p == 2 || p == 3 || p == 5 || p == 7){
            for (q = 1; q <= 5; q++){
                long long mod = 1;
                for (int i = 0; i < q; i++) mod *= p;

                clr(dp);
                for (n = 0; n < 101; n++){
                    for (k = 0; k <= n; k++){
                        if (k == 0 || k == n) dp[n][k] = 1;
                        else dp[n][k] = (dp[n - 1][k - 1] + dp[n - 1][k])
% mod;
                    }
                }

                for (n = 0; n < 101; n++){
                    for (k = 0; k < 101; k++){
                        if (dp[n][k] != bin::binomial(n, k, p, q)){
                            printf("%lld %lld = %lld %lld\n", n, k, p, q);
                            dbg(dp[n][k]);
                            return 0;
                        }
                    }
                }
            }
        }
    }

    for (m = 1; m <= 1000; m++){
        long long mod = m;
        clr(dp);
        for (n = 0; n < 101; n++){
            for (k = 0; k <= n; k++){
                if (k == 0 || k == n) dp[n][k] = 1 % mod;
                else dp[n][k] = (dp[n - 1][k - 1] + dp[n - 1][k]) % mod;
            }
        }

        for (n = 0; n < 101; n++){
            for (k = 0; k < 101; k++){
                if (dp[n][k] != bin::binomial(n, k, mod)){
                    printf("%lld %lld = %lld\n", n, k, mod);
                    dbg(dp[n][k]);
```

```
                    return 0;
                }
            }
        }
    }



    n = 1e9 - 10, k = (n / 3);
    p = 2, q = 16;

    dbg(bin::binomial(n, k, p, q));
    return 0;
}
```

Binomial from Factorial.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 1000010
#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

long long factorial[MAX];

long long expo(long long x, long long n){
    long long res = 1;

    while (n){
        if (n & 1LL) res = (res * x) % MOD;
        x = (x * x) % MOD;
        n >>= 1LL;
    }

    return (res % MOD);
}

long long binomial(int n, int k){
    long long x = factorial[n];
    long long y = (factorial[k] * factorial[n - k]) % MOD;
    long long res = (x * expo(y, MOD - 2)) % MOD;
    return res;
}

int main(){
    int i, n, k;

    factorial[0] = 1LL;
    for (i = 1; i < MAX; i++) factorial[i] = (factorial[i - 1] * i) % MOD;

    while (scanf("%d %d", &n, &k) != EOF){
```

```
        long long res = binomial(n, k);
        printf("%lld\n", res);
    }
    return 0;
}
```

Binomial from Prime Factorization.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 1000010
#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

bool prime[MAX];
int p, len, P[MAX], counter[MAX];

void Sieve(){
    clr(prime);
    int i, j, d;
    const int sqr = ;

    prime[2] = true;
    for (i = 3; i < MAX; i++, i++) prime[i] = true;

    for (i = 3; i < sqr;){
        d = i << 1;
        for (j = (i * i); j < MAX; j += d) prime[j] = false;

        i++, i++;
        while (!prime[i]) i++, i++;
    }

    p = 0;
    for (i = 0; i < MAX; i++){
        if (prime[i]) P[p++] = i;
    }
}

void factorize(int n, bool add){
    int i, j, c, x;

    prime[1] = true;
    for (i = 0; i < p; i++){
        if (prime[n]) break;

        c = 0, x = P[i];
        while ((n % x) == 0){
            n /= x;
            c++;
        }
```

```
        if (add) counter[x] += c;
        else counter[x] -= c;
    }

    if (n > 1){
        if (add) counter[n]++;
        else counter[n]--;
    }
}

long long expo(long long x, long long n){
    long long res = 1;

    while (n){
        if (n & 1LL) res = (res * x) % MOD;
        x = (x * x) % MOD;
        n >>= 1LL;
    }

    return (res % MOD);
}

long long binomial(int n, int k){
    int i, x;
    clr(counter);

    if ((n - k) > k) k = (n - k);
    for (i = k + 1; i <= n; i++) factorize(i, true);
    for (i = 2; i <= (n - k); i++) factorize(i, false);

    long long res = 1;
    for (i = 0; i < p; i++){
        x = P[i];
        if (x > n) break;
        if (counter[x]) res = (res * expo(x, counter[x])) % MOD;
    }

    return res;
}

int main(){
    Sieve();
    int n, k;

    while (scanf("%d %d", &n, &k) != EOF){
        clr(counter);
        long long res = binomial(n, k);
        printf("%lld\n", res);
    }
    return 0;
}
```

Bipartite Matching.c

```
-------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 1010
#define clr(ar) (memset(ar, 0, sizeof(ar)))
#define read() freopen("lol.txt", "r", stdin)

bool visited[MAX];
int t, line, n, m, ar[MAX][MAX], adj[MAX][MAX], left[MAX], right[MAX],
len[MAX];

bool bipartite_matching(int u){
   int v = 0, j = 0;
   for (j = 0; j < len[u]; j++){
       v = adj[u][j];
       if (visited[v]) continue;
       visited[v] = true;

       if (right[v] == -1 || bipartite_matching(right[v])){
           left[u] = v;
           right[v] = u;
           return true;
       }
   }
   return false;
}

int max_matching(){
   memset(left, -1, sizeof(left));
   memset(right, -1, sizeof(right));

   int i, counter = 0;
   for (i = 0; i < n; i++){
       clr(visited);
       if (bipartite_matching(i)) counter++;
   }
   return counter;
}

int main(){
   int a, b;
   while (scanf("%d %d", &n, &m) != EOF){
       clr(len);
       while (m--){
           scanf("%d %d", &a, &b);
           adj[a][len[a]++] = b;
       }
   }
   return 0;
}
```

Bit Hacks.c

```c
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

unsigned int reverse_bits(unsigned int v){
   v = ((v >> 1) & 0x55555555) | ((v & 0x55555555) << 1);
   v = ((v >> 2) & 0x33333333) | ((v & 0x33333333) << 2);
   v = ((v >> 4) & 0x0F0F0F0F) | ((v & 0x0F0F0F0F) << 4);
   v = ((v >> 8) & 0x00FF00FF) | ((v & 0x00FF00FF) << 8);
   return ((v >> 16) | (v << 16));
}

/// Returns i if x = 2^i and 0 otherwise
int bitscan(unsigned int x){
   __asm__ volatile("bsf %0, %0" : "=r" (x) : "0" (x));
   return x;
}

/// Returns next number with same number of 1 bits
unsigned int next_combination(unsigned int x){
   unsigned int y = x & -x;
   x += y;
   unsigned int z = x & -x;
   z -= y;
   z = z >> bitscan(z & -z);
   return x | (z >> 1);
}

int main(){
}
```

Bit String LCS.c

```c
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

bool flag[MAX];
char A[MAX], B[MAX], S[2][MAX];

int lcs(char* A, char* B){
   int i, j, k, v, n, m, res = 0;
   unsigned long long x, t, q, y, mask[128] = {0};

   clr(flag);
```

```
    for (n = 0; A[n]; n++) S[0][n] = A[n];
    for (m = 0; B[m]; m++) S[1][m] = B[m];

    for (i = 0; (i * 64) < m; i++){
        clr(mask);
        for (k = 0; k < 64 && (i * 64 + k) < m; k++){
            mask[S[1][i * 64 + k]] |= (1ULL << k);
        }

        for (j = 0, x = 0; j < n; j++){
            t = mask[S[0][j]] & ~x;
            x |= t, v = flag[j];
            q = x - (t << 1) - v, y = (q & ~x) | t;
            flag[j] = y >> 63, x &= ~(y << 1);
            if (v) x &= ~1ULL;
        }
        res += __builtin_popcountll(x);
    }
    return res;
}

int main(){
    int i, j, k, n, m;
    n = MAX - 10, m = MAX - 10;
    for (i = 0; i < n; i++) A[i] = (rand() % 26) + 'a';
    for (i = 0; i < m; i++) B[i] = (rand() % 26) + 'a';
    A[n] = B[m] = 0;

    clock_t start = clock();
    printf("%d\n", lcs(A, B));

    printf("%0.6f\n", (clock() - start) / (1.0 * CLOCKS_PER_SEC));
    return 0;
}

Bitset Iteration.c
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

char bithash[67];

void init(){
    int i;
    for (i = 0; i < 64; i++) bithash[(1ULL << i) % 67] = i;
}

void iterate(unsigned long long x){
    while (x){
        unsigned long long y = (x & -x);
```

```
        int i = bithash[y % 67];
        x ^= y;
        printf("%d\n", i);
    }
}

int main(){
    init();
    unsigned long long x = 0b1000000010110001001001010010101010010100;
    iterate(x);
    return 0;
}
```

Blocks DP.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

bool visited[200][200][200];
int t, n, ar[200], dp[200][200][200];

int F(int i, int j, int k){
    if (i > j) return 0;
    if (visited[i][j][k]) return dp[i][j][k];

    int l, res = 0;
    res = ((k + 1) * (k + 1)) + F(i + 1, j, 0);
    for (l = i + 1; l <= j; l++){
        if (ar[i] == ar[l]){
            int x = F(i + 1, l - 1, 0) + F(l, j, k + 1);
            if (x > res) res = x;
        }
    }

    visited[i][j][k] = true;
    return (dp[i][j][k] = res);
}

int main(){
    int T = 0, i, j;

    scanf("%d", &t);
    while (t--) {
        scanf("%d", &n);
        for (i = 0; i < n; i++) scanf("%d", &ar[i]);

        clr(visited);
        int res = F(0, (n - 1), 0);
        printf("Case %d: %d\n", ++T, res);
    }
```

```
   return 0;
}
```

Brent Pollard Rho OP 2.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

namespace ftor{
   /// Note use only for factorization
   #define MLIM 1000 /// Used in Brent Pollard Rho
   #define PLIM 10  /// Miller-Rabin Test depth
   #define MAX 10000 /// Small primes generated in this range
   #define SQR 101 /// Square root of MAX

   int p, P[MAX];
   bitset <MAX> prime;
   tr1::unordered_map <long long, long long> mp;

   inline void Sieve(){
       int i, j, d;
       prime.reset();
       prime[2] = true;
       for (i = 3; i < MAX; i++, i++) prime[i] = true;

       for (i = 3; i < SQR;){
           d = i << 1;
           for (j = (i * i); j < MAX; j += d) prime[j] = false;

           do{
               i++;
           } while (!prime[++i]);
       }

       p = 0;
       for (i = 0; i < MAX; i++){
           if (prime[i]) P[p++] = i;
       }
   }

   inline long long mul(long long a, long long b, long long m, long
double fuck) {
      long long c = (long long)(fuck * a * b);
      a *= b;
      a -= c * m;
      if (a >= m) a -= m;
      if (a < 0) a += m;
      return a;
   }
```

```
inline long long expo(long long x, long long n, long long m, long
double fuck){
    long long res = 1;

    while (n){
        if (n & 1) res = mul(res, x, m, fuck);
        x = mul(x, x, m, fuck);
        n >>= 1;
    }

    return res;
}

inline bool miller_rabin(long long p, long double fuck, int lim){
    long long a, s, m, x, y;
    s = p - 1, y = p - 1;
    while (!(s & 1)) s >>= 1;

    while (lim--){
        x = s;
        a = (((rand() << 15) ^ rand()) % y) + 1;
        m = expo(a, x, p, fuck);

        while ((x != y) && (m != 1) && (m != y)){
            m = mul(m, m, p, fuck);
            x <<= 1;
        }
        if ((m != y) && !(x & 1)) return false;
    }

    return true;
}

inline unsigned long long gcd(unsigned long long u, unsigned long long
v){
    if (!u) return v;
    if (!v) return u;
    if (u == 1 || v == 1) return 1;

    int shift = __builtin_ctzll(u | v);
    u >>= __builtin_ctzll(u);
    do{
        v >>= __builtin_ctzll(v);
        if (u > v)
            v ^= u ^= v ^= u;
        v = v - u;
    } while (v);

    return u << shift;
}

inline long long brent_pollard_rho(long long n){
    long double fuck = (long double)1 / n;
```

```
if (miller_rabin(n, fuck, PLIM)) return n;

const long long m = MLIM;
long long i, k, a, x, y, ys, r, q, g;
g = mp[n];
if (g) return g;

do{
    a = ((rand() << 15) ^ rand()) % n;
}
while (!a || a == (n - 2));

r = 1, q = 1;
y = ((rand() << 15) ^ rand()) % n;

do{
    x = y;
    for (i = 0; i < r; i++){
        y = mul(y, y, n, fuck);
        y += a;
        if (y < a) y += (ULLONG_MAX - n) + 1;
        y %= n;
    }

    k = 0;
    do{
        for (i = 0; (i < m) && (i < (r - k)); i++){
            ys = y;
            y = mul(y, y, n, fuck) + a;
            if (y < a) y += (ULLONG_MAX - n) + 1;
            y %= n;
            q = mul(q, abs(x - y), n, fuck);
        }

        g = gcd(q, n);
        k += m;

    }
    while ((k < r) && (g == 1));

    r <<= 1;
}
while (g == 1);

if (g == n){
    do{
        ys = mul(ys, ys, n, fuck) + a;
        if (ys < a) ys += (ULLONG_MAX - n) + 1;
        ys %= n;
        g = gcd(abs(x - ys), n);
    }
    while (g == 1);
}
```

```
    return (mp[n] = g);
}

vector <long long> factorize(long long n){
    int i, d, len;
    long long m, k, x;
    vector <long long> v, factors;

    for (i = 0; i < p; i++){
        while (!(n % P[i])){
            n /= P[i];
            v.push_back(P[i]);
        }
    }
    if (n == 1) return v;

    x = brent_pollard_rho(n);
    factors.push_back(x);
    factors.push_back(n / x);
    len = factors.size();

    do{
        m = factors[len - 1];
        factors.pop_back(), len--;

        if (m > 1){
            if (miller_rabin(m, (long double)1 / m, PLIM)){
                v.push_back(m);
                for (i = 0; i < len; i++){
                    k = factors[i];
                    while (!(k % m)){
                        k /= m;
                        v.push_back(m);
                    }
                    factors[i] = k;
                }
            }
            else{
                x = brent_pollard_rho(m);
                factors.push_back(x);
                factors.push_back(m / x);
                len += 2;
            }
        }
    }
    while (len);

    sort(v.begin(), v.end());
    return v;
}

void init(){
    Sieve();
    mp.clear();
```

```
        srand(time(0));
    }
}

int main(){
    ftor::init();
    vector <long long> v = ftor::factorize(210);
    return 0;
}
```

Brent Pollard Rho OP.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

namespace ftor{
    /// Note use only for factorization
    #define MLIM 1000 /// Used in Brent Pollard Rho
    #define PLIM 10  /// Miller-Rabin Test depth
    #define MAX 10000 /// Small primes generated in this range
    #define SQR 101 /// Square root of MAX

    int p, P[MAX];
    bitset <MAX> prime;
    tr1::unordered_map <long long, long long> mp;

    inline void Sieve(){
        int i, j, d;
        prime.reset();
        prime[2] = true;
        for (i = 3; i < MAX; i++, i++) prime[i] = true;

        for (i = 3; i < SQR;){
            d = i << 1;
            for (j = (i * i); j < MAX; j += d) prime[j] = false;

            do{
                i++;
            } while (!prime[++i]);
        }

        p = 0;
        for (i = 0; i < MAX; i++){
            if (prime[i]) P[p++] = i;
        }
    }

    inline long long mul(long long a, long long b, long long MOD) {
        long double res = a;
```

```
        res *= b;
        long long c = (long long)(res / MOD);
        a *= b;
        a -= c * MOD;
        if (a >= MOD) a -= MOD;
        if (a < 0) a += MOD;
        return a;
    }

    inline long long expo(long long x, long long n, long long m){
        long long res = 1;

        while (n){
            if (n & 1) res = mul(res, x, m);
            x = mul(x, x, m);
            n >>= 1;
        }

        return (res % m);
    }

    inline bool miller_rabin(long long p, int lim){
        long long a, s, m, x, y;
        s = p - 1, y = p - 1;
        while (!(s & 1)) s >>= 1;

        while (lim--){
            x = s;
            a = (((rand() << 15) ^ rand()) % y) + 1;
            m = expo(a, x, p);

            while ((x != y) && (m != 1) && (m != y)){
                m = mul(m, m, p);
                x <<= 1;
            }
            if ((m != y) && !(x & 1)) return false;
        }

        return true;
    }

    inline unsigned long long gcd(unsigned long long u, unsigned long long
v){
        if (!u) return v;
        if (!v) return u;
        if (u == 1 || v == 1) return 1;

        int shift = __builtin_ctzll(u | v);
        u >>= __builtin_ctzll(u);
        do{
            v >>= __builtin_ctzll(v);
            if (u > v)
                v ^= u ^= v ^= u;
            v = v - u;
```

```
    } while (v);

    return u << shift;
}

inline long long brent_pollard_rho(long long n){
    if (miller_rabin(n, PLIM)) return n;

    const long long m = MLIM;
    long long i, k, a, x, y, ys, r, q, g;
    g = mp[n];
    if (g) return g;

    do{
        a = ((rand() << 15) ^ rand()) % n;
    }
    while (!a || a == (n - 2));

    r = 1, q = 1;
    y = ((rand() << 15) ^ rand()) % n;

    do{
        x = y;
        for (i = 0; i < r; i++){
            y = mul(y, y, n);
            y += a;
            if (y < a) y += (ULLONG_MAX - n) + 1;
            y %= n;
        }

        k = 0;
        do{
            for (i = 0; (i < m) && (i < (r - k)); i++){
                ys = y;
                y = mul(y, y, n) + a;
                if (y < a) y += (ULLONG_MAX - n) + 1;
                y %= n;
                q = mul(q, abs(x - y), n);
            }

            g = gcd(q, n);
            k += m;

        }
        while ((k < r) && (g == 1));

        r <<= 1;
    }
    while (g == 1);

    if (g == n){
        do{
            ys = mul(ys, ys, n) + a;
            if (ys < a) ys += (ULLONG_MAX - n) + 1;
```

```
            ys %= n;
            g = gcd(abs(x - ys), n);
        }
        while (g == 1);
    }

    return (mp[n] = g);
}

vector <long long> factorize(long long n){
    int i, d, len;
    long long m, k, x;
    vector <long long> v, factors;

    for (i = 0; i < p; i++){
        while (!(n % P[i])){
            n /= P[i];
            v.push_back(P[i]);
        }
    }
    if (n == 1) return v;

    x = brent_pollard_rho(n);
    factors.push_back(x);
    factors.push_back(n / x);
    len = factors.size();

    do{
        m = factors[len - 1];
        factors.pop_back(), len--;

        if (m > 1){
            if (miller_rabin(m, PLIM)){
                v.push_back(m);
                for (i = 0; i < len; i++){
                    k = factors[i];
                    while (!(k % m)){
                        k /= m;
                        v.push_back(m);
                    }
                    factors[i] = k;
                }
            }
            else{
                x = brent_pollard_rho(m);
                factors.push_back(x);
                factors.push_back(m / x);
                len += 2;
            }
        }
    }
    while (len);

    sort(v.begin(), v.end());
```

```
        return v;
    }

    void init(){
        Sieve();
        mp.clear();
        srand(time(0));
    }
}

int main(){
    ftor::init();
    vector <long long> v = ftor::factorize(210);
    return 0;
}
```

Brent Pollard Rho.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

namespace ftor{
    /// Note use only for factorization
    #define MLIM 1000 /// Used in Brent Pollard Rho
    #define PLIM 10  /// Miller-Rabin Test depth
    #define MAX 10000 /// Small primes generated in this range
    #define SQR 101 /// Square root of MAX

    const long long LIM = LLONG_MAX;

    int p, P[MAX];
    bitset <MAX> prime;
    tr1::unordered_map <long long, long long> mp;

    void Sieve(){
        int i, j, d;
        prime.reset();
        prime[2] = true;
        for (i = 3; i < MAX; i++, i++) prime[i] = true;

        for (i = 3; i < SQR;){
            d = i << 1;
            for (j = (i * i); j < MAX; j += d) prime[j] = false;

            do{
                i++;
            } while (!prime[++i]);
        }
```

```
        p = 0;
        for (i = 0; i < MAX; i++){
            if (prime[i]) P[p++] = i;
        }
    }

    unsigned long long mul(unsigned long long a, unsigned long long b,
unsigned long long m){
        a %= m, b %= m;
        if (a > b) swap(a, b);
        if (!a) return 0;
        if (a < (LIM / b)) return ((a * b) % m);

        unsigned long long res = 0;
        int x, k = min(30, __builtin_clzll(m) - 1);
        int bitmask = (1 << k) - 1;

        while (a > 0){
            x = a & bitmask;
            res = (res + (b * x)) % m;
            a >>= k;
            b = (b << k) % m;
        }
        return res;
    }

    long long expo(long long x, long long n, long long m){
        long long res = 1;

        while (n){
            if (n & 1) res = mul(res, x, m);
            x = mul(x, x, m);
            n >>= 1;
        }

        return (res % m);
    }

    bool miller_rabin(long long p, int lim){
        long long a, s, m, x, y;
        s = p - 1, y = p - 1;
        while (!(s & 1)) s >>= 1;

        while (lim--){
            x = s;
            a = (((rand() << 15) ^ rand()) % y) + 1;
            m = expo(a, x, p);

            while ((x != y) && (m != 1) && (m != y)){
                m = mul(m, m, p);
                x <<= 1;
            }
            if ((m != y) && !(x & 1)) return false;
        }
```

```
    return true;
}

unsigned long long gcd(unsigned long long u, unsigned long long v){
    if (!u) return v;
    if (!v) return u;
    if (u == 1 || v == 1) return 1;

    int shift = __builtin_ctzll(u | v);
    u >>= __builtin_ctzll(u);
    do{
        v >>= __builtin_ctzll(v);
        if (u > v)
            v ^= u ^= v ^= u;
        v = v - u;
    } while (v);

    return u << shift;
}

long long brent_pollard_rho(long long n){
    if (miller_rabin(n, PLIM)) return n;

    const long long m = MLIM;
    long long i, k, a, x, y, ys, r, q, g;
    g = mp[n];
    if (g) return g;

    do{
        a = ((rand() << 15) ^ rand()) % n;
    }
    while (!a || a == (n - 2));

    r = 1, q = 1;
    y = ((rand() << 15) ^ rand()) % n;

    do{
        x = y;
        for (i = 0; i < r; i++){
            y = mul(y, y, n);
            y += a;
            if (y < a) y += (ULLONG_MAX - n) + 1;
            y %= n;
        }

        k = 0;
        do{
            for (i = 0; (i < m) && (i < (r - k)); i++){
                ys = y;
                y = mul(y, y, n) + a;
                if (y < a) y += (ULLONG_MAX - n) + 1;
                y %= n;
                q = mul(q, abs(x - y), n);
```

```
            }

            g = gcd(q, n);
            k += m;

        }
        while ((k < r) && (g == 1));

        r <<= 1;
    }
    while (g == 1);

    if (g == n){
        do{
            ys = mul(ys, ys, n) + a;
            if (ys < a) ys += (ULLONG_MAX - n) + 1;
            ys %= n;
            g = gcd(abs(x - ys), n);
        }
        while (g == 1);
    }

    return (mp[n] = g);
}

vector <long long> factorize(long long n){
    int i, d, len;
    long long m, k, x;
    vector <long long> v, factors;

    for (i = 0; i < p; i++){
        while (!(n % P[i])){
            n /= P[i];
            v.push_back(P[i]);
        }
    }
    if (n == 1) return v;

    x = brent_pollard_rho(n);
    factors.push_back(x);
    factors.push_back(n / x);
    len = factors.size();

    do{
        m = factors[len - 1];
        factors.pop_back(), len--;

        if (m > 1){
            if (miller_rabin(m, PLIM)){
                v.push_back(m);
                for (i = 0; i < len; i++){
                    k = factors[i];
                    while (!(k % m)){
                        k /= m;
```

```
                           v.push_back(m);
                       }
                       factors[i] = k;
                   }
               }
               else{
                   x = brent_pollard_rho(m);
                   factors.push_back(x);
                   factors.push_back(m / x);
                   len += 2;
               }
           }
       }
       while (len);

       sort(v.begin(), v.end());
       return v;
   }

   void init(){
       Sieve();
       mp.clear();
       srand(time(0));
   }
}

int main(){
   ftor::init();
   vector <long long> v = ftor::factorize(210);
   return 0;
}
```

Bridge Extended.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/// Note: 0-based indexing for graphs

struct bridge{
   int u, v, w; /// Bridge from node u to v with cost w
   int f, s; /// Number of components in the first and second graph if
bridge is disconnected

   bridge(){
   }

   bridge(int a, int b, int c, int d, int e){
```

```
        if (a > b) swap(a, b); /// Lower node first for convenience
        u = a, v = b, w = c, f = d, s = e;
    }
};

namespace br{
    typedef pair<int, int> Pair;

    bool visited[MAX];
    vector <bridge> B;
    vector <Pair> adj[MAX];
    tr1::unordered_set <long long> S;
    int n, r, dt, discover[MAX], low[MAX], cmp[MAX], num[MAX];

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

    void dfs(int i, int p){ /// hash = 60375
        visited[i] = true;
        discover[i] = low[i] = ++dt;
        int j, x, l, children = 0, len = adj[i].size();

        for (j = 0; j < len; j++){
            x = adj[i][j].first;
            if (!visited[x]){
                dfs(x, i);
                children++;
                low[i] = min(low[i], low[x]);

                if (low[x] > discover[i]){
                    if (!(j && adj[i][j - 1].first == x) || ( (j + 1) <
len && adj[i][j + 1].first == x) ){ /// Handling multi-edge
                        l = dt - discover[x] + 1;
                        B.push_back( bridge(i, x, adj[i][j].second, l,
cmp[i] - l) );
                    }
                }
            }
            else if (x != p) low[i] = min(low[i], discover[x]);
        }
    }

    void dfs(int i){
        low[dt++] = i;
        visited[i] = true;

        int j, x, len = adj[i].size();
        for (j = 0; j < len; j++){
            x = adj[i][j].first;
            if (!visited[x]){
                dfs(x);
            }
```

```
        }
    }

    /// Adds undirected edge from a to b with cost c or edge index number
i
    void add(int a, int b, int c){
        adj[a].push_back(Pair(b, c));
        adj[b].push_back(Pair(a, c));
    }

    void FindBridge(){
        int i, j;
        B.clear();
        clr(visited);
        for (i = 0; i < n; i++) sort(adj[i].begin(), adj[i].end()); /// To
handle multi-edges

        for (i = 0; i < n; i++){
            if (!visited[i]){
                dt = 0;
                dfs(i);
                for (j = 0; j < dt; j++) cmp[low[j]] = dt;
            }
        }

        clr(visited);
        for (i = 0; i < n; i++){
            if (!visited[i]){
                dt = 0;
                dfs(i, -1);
            }
        }
    }

    long long hashval(long long u, long long v){
        return (u * MAX) + v;
    }

    void dfsnum(int i){
        num[i] = r;
        visited[i] = true;

        int j, x, len = adj[i].size();
        for (j = 0; j < len; j++){
            x = adj[i][j].first;
            if (!visited[x] && !S.count(hashval(i, x))){
                dfsnum(x);
            }
        }
    }

    /// Call FindBridge before
    void BridgeTree(){
        S.clear();
```

```
        int i, j, x, u, v, len = B.size();

        for (i = 0; i < len; i++){
            S.insert(hashval(B[i].u, B[i].v));
            S.insert(hashval(B[i].v, B[i].u));
        }

        r = 0; /// Number of nodes in bridge tree
        clr(visited);
        for (i = 0; i < n; i++){
            if (!visited[i]){
                dfsnum(i);
                r++;
            }
        }

        for (i = 0; i < len; i++){
            u = B[i].u, v = B[i].v;
            /// Build the bridge tree here accordingly
        }
    }
}

int main(){
}
```

Bridge.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/// Note: 0-based indexing for graphs
namespace fuck{
    typedef pair<int, int> Pair;

    vector <Pair> bridge;
    vector <Pair> adj[MAX];
    bool cut[MAX], visited[MAX];
    int n, disc_t, discover[MAX], low[MAX];

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

    void dfs(int i, int p){
        visited[i] = true;
        discover[i] = low[i] = ++disc_t;
```

```
        int j, x, cmp, children = 0, len = adj[i].size();

        for (j = 0; j < len; j++){
            x = adj[i][j].first;
            if (!visited[x]){
                children++;
                dfs(x, i);
                low[i] = min(low[i], low[x]);

                if (low[x] > discover[i]){
                    int cmp = -1; /// cmp used to check for multi-edge
                    if (j) cmp = adj[i][j - 1].first;
                    if (cmp != x && (j + 1) < len) cmp = adj[i][j +
1].first;
                    if (cmp != x) bridge.push_back(Pair(min(i, x), max(i,
x))); /// lower node comes first in bridge representation
                }
            }
            else if (x != p) low[i] = min(low[i], discover[x]);
        }
    }

    /// Adds undirected edge from a to b with cost or edge index number i
    /// Note that i is optional, it's not needed to find bridges

    void add(int a, int b, int i){
        adj[a].push_back(Pair(b, i));
        adj[b].push_back(Pair(a, i));
    }

    void FindBridge(){
        int i, j;
        bridge.clear();
        clr(cut), clr(visited);
        for (i = 0; i < n; i++) sort(adj[i].begin(), adj[i].end()); /// To
handle multi-edges

        for (i = 0; i < n; i++){
            if (!visited[i]){
                disc_t = 0;
                dfs(i, -1);
            }
        }
        /// bridge vector contains all the bridges as pairs now
    }
}

int main(){
}
```

CKY Algorithm.c

The Cocke\96Younger\96Kasami algorithm (alternatively called CYK, or CKY) is a parsing algorithm for context-free grammars,

named after its inventors, John Cocke, Daniel Younger and Tadao Kasami.
It employs bottom-up parsing and dynamic programming.

The standard version of CYK operates only on context-free grammars given in Chomsky normal form (CNF).
However any context-free grammar may be transformed to a CNF grammar expressing the same language.

The importance of the CYK algorithm stems from its high efficiency in certain situations. Using Landau symbols,
the worst case running time of CYK is O(N^3 * |G|), where N is the length of the parsed string and |G| is the
size of the CNF grammar G. This makes it one of the most efficient parsing algorithms in terms of worst-case
asymptotic complexity, although other algorithms exist with better average running time in many practical scenarios.

Pseudocode:

```
let the input be a string S consisting of n characters: a1 ... an.
let the grammar contain r nonterminal symbols R1 ... Rr.
This grammar contains the subset Rs which is the set of start symbols.
let P[n,n,r] be an array of booleans. Initialize all elements of P to
false.
for each i = 1 to n
 for each unit production Rj -> ai
   set P[1,i,j] = true
for each i = 2 to n -- Length of span
 for each j = 1 to n-i+1 -- Start of span
   for each k = 1 to i-1 -- Partition of span
     for each production RA -> RB RC
       if P[k,j,B] and P[i-k,j+k,C] then set P[i,j,A] = true
if any of P[n,1,x] is true (x is iterated over the set s, where s are all
the indices for Rs) then
 S is member of language
else
 S is not member of language
```

Cartesian Tree.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/*** Note that duplicates are discarded by the unordered_set ***/

class Treap{
   public:

   struct Node{
```

```
        Node *l, *r;
        int val, counter;
        unsigned int priority;

        Node(){
            l = r = 0;
        }

        Node(int x){
            l = r = 0;
            counter = 1, val = x;
            priority = rand();
            if (priority < 32768) priority = (priority << 17) ^ rand();
        }

        ~Node(){
            delete l;
            delete r;
        }

        void update(){
            counter = 1;
            if (l) counter += l->counter;
            if (r) counter += r->counter;
        }
    };

    /*** Merge two treaps in O(log n), Assumes l and r are in the
appropriate order
         All values in the first are less than the values at the second
    ***/
    Node* merge(Node* l, Node* r){
        if (!l) return r;
        if (!r) return l;

        if (l->priority < r->priority){
            l->r = merge(l->r, r);
            l->update();
            return l;
        }
        else{
            r->l = merge(l, r->l);
            r->update();
            return r;
        }
    }

    /*** Divides the tree T into two trees L and R so that L contains all
elements that
         are smaller than x, and R contains all elements that are equal to
or larger than x.
    ***/
    void split(Node* cur, Node* &l, Node* &r, int x){
        l = r = 0;
```

```
        if (!cur) return;

        if (cur->val < x){
            split(cur->r, cur->r, r, x);
            l = cur;
        }
        else{
            split(cur->l, l, cur->l, x);
            r = cur;
        }

        cur->update();
    }

    Node* root;
    tr1::unordered_set <int> S; /*** Remove if no duplicates are
guaranteed ***/

    Treap(){
        root = 0;
        S.clear();
        srand(time(0));
    }

    ~Treap(){
        delete root;
    }

    bool insert(int x){
        if (S.count(x)) return false;

        Node *l, *r;
        split(root, l, r, x);
        root = merge(l, merge(new Node(x), r));
        S.insert(x);
        return true;
    }

    bool erase(int x){
        if (!S.count(x)) return false;

        Node *l, *r, *m;
        split(root, l, m, x);
        split(m, m, r, x + 1);

        if (m && m->counter == 1 && m->val == x){
            delete m;
            root = merge(l, r);
            S.erase(x);
            return true;
        }
        else return false;
    }
```

```
    int size(){
        if (root) return root->counter;
        return 0;
    }

    /*** Returns the k'th smallest element of the treap in 1-based index,
-1 on failure ***/
    int kth(int k){
        if ((k < 1) || (k > size())) return -1;

        Node *l, *r, *cur = root;
        for (; ;){
            l = cur->l, r = cur->r;
            if (l){
                if (k <= l->counter) cur = l;
                else if ((l->counter + 1) == k) return cur->val;
                else cur = r, k -= (l->counter + 1);
            }
            else{
                if (k == 1) return (cur->val);
                else cur = r, k--;
            }
        }
    }

    /*** Returns the count of values strictly less than x ***/
    int count(int x){
        int res = 0;
        Node *l, *r, *cur = root;

        while (cur){
            l = cur->l, r = cur->r;
            if (cur->val < x) res++;
            if (x < cur->val) cur = l;
            else{
                cur = r;
                if (l) res += l->counter;
            }
        }
        return res;
    }
};

int main(){
    char str[10];
    int n, i, j, x, res;

    scanf("%d", &n);
    Treap T = Treap();
    while (n--){
        scanf("%s %d", str, &x);
        if (str[0] == 'I') T.insert(x);
        if (str[0] == 'D') T.erase(x);
        if (str[0] == 'C') printf("%d\n", T.count(x));
```

```
        if (str[0] == 'K'){
            res = T.kth(x);
            if (res == -1) puts("invalid");
            else printf("%d\n", res);
        }
    }
    return 0;
}
```

Chinese Remainder Theorem.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

long long extended_gcd(long long a, long long b, long long& x, long long&
y){
    /// Bezout's identity, ax + by = gcd(a,b)

    if (!b){
        y = 0, x = 1;
        return a;
    }

    long long g = extended_gcd(b, a % b, y, x);
    y -= ((a / b) * x);
    return g;
}

long long mod_inverse(long long a, long long m){
    /// inverse exists if and only if a and m are co-prime

    long long x, y, inv;
    extended_gcd(a, m, x, y);
    inv = (x + m) % m;
    return inv;
}

long long CRT(int n, long long* ar, long long* mods){
    /// finds the unique solution x modulo M (product of mods) for which x
% mods[i] = ar[i]
    /// mods must be pairwise co-prime

    int i, j;
    long long x, y, res = 0, M = 1;

    for (i = 0; i < n; i++) M *= mods[i];
    for (i = 0; i < n; i++){
        x = M / mods[i];
        y = mod_inverse(x, mods[i]);
```

```
        res = (res + (((x * ar[i]) % M) * y)) % M;
    }

    return res;
}

int main(){
    long long mods[] = {3, 5, 7};
    long long ar[] = {2, 3, 2};
    long long res = CRT(3, ar, mods);
    dbg(res);
}
```

Circle Circle Intersection Area.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define sqr(x) ((x) * (x))
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

struct Point{
    double x, y;

    Point() {
    }

    Point(double a, double b){
        x = a, y = b;
    }
};

struct Circle{
    Point centre;
    double radius;

    Circle(){
    }

    Circle(double a, double b, double c){
        radius = c;
        centre = Point(a, b);
    }

    Circle(struct Point c, double r){
        centre = c;
        radius = r;
    }
};
```

```
const double pi = 2.0 * acos(0.0);

double dis(Point A, Point B){
   double x = (A.x - B.x);
   double y = (A.y - B.y);
   double res = sqrt((x * x) + (y * y));
   return res;
}

double Area(Circle C){
   return (pi * sqr(C.radius));
}

double intersection(Circle A, Circle B){
   double x = A.radius + B.radius;
   double y = dis(A.centre, B.centre);
   if ((fabs(x - y) <= 1e-8) || (y > x)) return 0.0;

   double c = y;
   double x0 = A.centre.x, y0 = A.centre.y, r0 = A.radius;
   double x1 = B.centre.x, y1 = B.centre.y, r1 = B.radius;

   x = (sqr(r1) + sqr(c) - sqr(r0)) / (2.0 * r1 * c);
   double CBD = acos(x) * 2.0;
   y = (sqr(r0) + sqr(c) - sqr(r1)) / (2.0 * r0 * c);
   double CAD = acos(y) * 2.0;


   double res = (0.5 * CBD * sqr(r1)) - (0.5 * sqr(r1) * sin(CBD)) + (0.5
* CAD * sqr(r0)) - (0.5 * sqr(r0) * sin(CAD));
   return res;
}

bool Inside(Circle A, Circle B){
   double x = dis(A.centre, B.centre) + B.radius;
   if ((fabs(x - A.radius) <= 1e-8) || (x < A.radius)) return true;
   return false;
}

int main(){
   double res;
   int T = 0, t, i, a, b, c;

   scanf("%d", &t);
   while (t--){
       scanf("%d %d %d", &a, &b, &c);
       Circle grass = Circle(a, b, c);
       scanf("%d %d %d", &a, &b, &c);
       Circle wolf = Circle(a, b, c);

       if (Inside(wolf, grass)) res = 0.0;
       else if (Inside(grass, wolf)) res = Area(grass) - Area(wolf);
       else{
           res = Area(grass) - intersection(grass, wolf);
```

```
        }

        printf("Case #%d: %0.12f\n", ++T, res + 1e-15);
    }
    return 0;
}
```

Code Hash.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

unsigned long long code_hash(int MOD = 1000003, int BASE = 1000000007){
/// hash = 124778
    freopen("copy.txt", "r", stdin);

    char str[1010];
    unsigned long long h = 0;

    while (scanf("%s", str) != EOF){
        int l = strlen(str);
        for (int j = 0; str[j]; j++){
            h = ((h * BASE) + str[j]) % MOD;
            assert(! ((j + 1) < l && str[j] == '/' && str[j + 1] == '/')
); /// Checking whether comments are removed
            assert(! ((j + 1) < l && str[j] == '/' && str[j + 1] == '*')
); /// Checking whether comments are removed
        }
    }
    return h;
}

int main(){
    printf("/// hash = %llu\n", code_hash());
    return 0;
}
```

Combsort.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int n, ar[MAX];
```

```
void combsort(int n, int* ar){
   if (n <= 1) return;
   int i, j, x, g = n, flag = 0;
   while ((g != 1) || flag){
       flag = 0;
       if (g != 1) g *= 0.77425;

       for (i = 0; (i + g) < n; i++){
           if (ar[i] > ar[i + g]){
               flag = 1;
               x = ar[i], ar[i] = ar[i + g], ar[i + g] = x;
           }
       }
   }
}

int main(){
   int i, j;
   n = MAX - 10;
   for (i = 0; i < n; i++) ar[i] = (rand() << 15 ^ rand()) % MAX;

   combsort(n, ar);
   for (i = 0; (i + 1) < n; i++){
       if (ar[i] > ar[i + 1]) puts("fail");
   }
   return 0;
}
```

Convex Hull.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

struct Point {
   long long x, y; /// x*x or y*y should fit long long because of cross()
function

   Point(){}
   Point(long long a, long long b){
       x = a, y = b;
   }

   inline bool operator < (const Point &p) const {
       return ((x < p.x) || (x == p.x && y < p.y));
   }
};

long long cross(const Point &O, const Point &A, const Point &B){
```

```
    return ((A.x - O.x) * (B.y - O.y)) - ((A.y - O.y) * (B.x - O.x));
}

bool isConvex(vector <Point> P){ /// Polygon P convex or not, P is given
in clock-wise of anti-clockwise order
    int n = P.size();
    ///sort(P.begin(), P.end());
    if (n <= 2) return false; /// Line or point is not convex

    n++, P.push_back(P[0]); /// Last point = first point
    bool flag = (cross(P[0], P[1], P[2]) > 0);
    for (int i = 1; (i + 1) < n; i++){
        bool cmp = (cross(P[i], P[i + 1], (i + 2) == n ? P[1] : P[i + 2])
> 0);
        if (cmp ^ flag) return false;
    }

    return true;
}

/// Convex hull using the monotone chain algorithm
vector <Point> convex_hull (vector<Point> P){
    int i, t, k = 0, n = P.size();
    vector<Point> H(n << 1);
    sort(P.begin(), P.end()); /// Can be converted to O(n) using radix
sort

    for (i = 0; i < n; i++) {
        while (k >= 2 && cross(H[k - 2], H[k - 1], P[i]) < 0) k--;
        H[k++] = P[i];
    }
    for (i = n - 2, t = k + 1; i >= 0; i--) {
        while (k >= t && cross(H[k - 2], H[k - 1], P[i]) < 0) k--;
        H[k++] = P[i];
    }

    H.resize(k - 1); /// The last point is the same as the first in this
implementation
    return H;
}

int main(){

}
```

Coordinate Compression.cpp

```
-------------------------------------------------
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))
```

```
using namespace std;

/// not necessarily sorted
void compress(int n, int* in, int* out){ /// 0 based index
   unordered_map <int, int> mp;
   for (int i = 0; i < n; i++) out[i] = mp.emplace(in[i],
mp.size()).first->second;
}

int main(){
   int data[] = {3, 10, 20, 15, 5, 2, 100, 10, 25, 2, 5};
   map <int, int> mp;
   int ar[100];

   for (int i = 0; i < 11; i++){
       ar[i] = mp.emplace(data[i], mp.size()).first->second;
       dbg(ar[i]);
   }
}
```

DSU On Subtree.cpp

```
/// Codeforces 601D
/// D[i] = number of distinct strings starting at vertex i and ending on
some vertex down subtree i

#include <bits/stdtr1c++.h>

#define MAX 300010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

const long long HMOD[] = {2078526727, 2117566807};
const long long BASE[] = {1572872831, 1971536491};

char str[MAX];
vector <int> adj[MAX];
int n, C[MAX], D[MAX], T[MAX];
set <long long> S[MAX];

void dfs(int i, int p, long long h1, long long h2){ /// hash = 400687
   h1 = (h1 * BASE[0] + str[i - 1] + 10007) % HMOD[0];
   h2 = (h2 * BASE[1] + str[i - 1] + 10007) % HMOD[1];

   int j, k, x, idx = 0, res = 0, len = adj[i].size();
   for (j = 0; j < len; j++){
       x = adj[i][j];
       if (x != p){
           dfs(x, i, h1, h2);
           if (D[x] > res) res = D[x], idx = x; /// update
       }
```

```
    }

    if (idx) T[i] = T[idx]; /// If maximum subtree child found, set root
to root of subtree
    for (j = 0; j < len; j++){
        x = adj[i][j];
        if (x != p && T[x] != T[i]){
            for (auto it: S[T[x]]) S[T[i]].insert(it); /// If not parent
and not maximum subtree, insert
            S[T[x]].clear(); /// Finally clear the remaining items since
not required anymore
        }
    }
    S[T[i]].insert((h1 << 31) ^ h2);
    D[i] = S[T[i]].size();
}

int main(){
    int i, j, k, l, a, b, c, x, res;

    scanf("%d", &n);
    for (i = 1; i <= n; i++) T[i] = i;
    for (i = 1; i <= n; i++) scanf("%d", &C[i]); /// Set parent[i] = i

    scanf("%s", str);
    for (i = 1; i < n; i++){
        scanf("%d %d", &a, &b);
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    dfs(1, 0, 0, 0);
    for (res = 0, c = 0, i = 1; i <= n; i++){
        x = C[i] + D[i];
        if (x > res) res = x, c = 1;
        else if (x == res) c++;
    }

    printf("%d\n%d\n", res, c);
    return 0;
}
```

DSU.cpp

```
--------------------------------------------------
#include <stdio.h>
#include <bits/stdtr1c++.h>

#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;
```

```
namespace dsu{
   int n, parent[MAX], counter[MAX];

   inline void init(int nodes){
       n = nodes;
       for (int i = 0; i <= n; i++){
           parent[i] = i;
           counter[i] = 1;
       }
   }

   inline int find_root(int i){
       while (i != parent[i]){
           parent[i] = parent[parent[i]];
           i = parent[i];
       }
       return parent[i];
   }

   inline void merge(int a, int b){
       int c = find_root(a), d = find_root(b);
       if (c != d){
           parent[c] = d;
           counter[d] += counter[c], counter[c] = 0;
       }
   }

   inline bool connected(int a, int b){
       int c = find_root(a), d = find_root(b);
       return (c == d);
   }

   inline int component_size(int i){
       int p = find_root(i);
       return counter[p];
   }
}

int main(){

}
```

Digits of Factorial.c

```
--------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

const double A = 4.0 * acos(0.0);
```

```
const double B = log10(exp(1.0));

long long digfact(long long n){
   if (n == 0 || n == 1) return 1;
   double x = ((0.5 * log(A * n)) + (n * log(n)) - n) * B;
   return ceil(x);
}

int main(){
   int t;
   long long n;

   scanf("%d", &t);
   while (t--){
       scanf("%lld", &n);
       printf("%lld\n", digfact(n));
   }
   return 0;
}
```

Dinic_s Algorithm (Edge List).cpp

```
#include <bits/stdtr1c++.h>

#define MAXN 30010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Dinic's algorithm for directed graphs (0 based index for graphs)
/// For undirected graphs, just add two directed edges

const long long INF = (~0ULL) >> 1;

namespace flow{
   struct Edge{
       int u, v;
       long long cap, flow;

       Edge(){}
       Edge(int a, int b, long long c, long long f){
           u = a, v = b, cap = c, flow = f;
       }
   };

   vector <int> adj[MAXN];
   vector <struct Edge> E;
   int n, s, t, ptr[MAXN], len[MAXN], dis[MAXN], Q[MAXN];

   inline void init(int nodes, int source, int sink){
       clr(len);
```

```
        E.clear();
        n = nodes, s = source, t = sink;
        for (int i = 0; i < MAXN; i++) adj[i].clear();
    }

    /// Adds a directed edge with capacity c
    inline void addEdge(int a, int b, long long c){
        adj[a].push_back(E.size());
        E.push_back(Edge(a, b, c, 0));
        len[a]++;
        adj[b].push_back(E.size());
        E.push_back(Edge(b, a, 0, 0));
        len[b]++;
    }

    inline bool bfs(){
        int i, j, k, id, f = 0, l = 0;
        memset(dis, -1, sizeof(dis[0]) * n);

        dis[s] = 0, Q[l++] = s;
        while (f < l && dis[t] == -1){
            i = Q[f++];
            for (k = 0; k < len[i]; k++){
                id = adj[i][k];
                if (dis[E[id].v] == -1 && E[id].flow < E[id].cap){
                    Q[l++] = E[id].v;
                    dis[E[id].v] = dis[i] + 1;
                }
            }
        }
        return (dis[t] != -1);
    }

    long long dfs(int i, long long f){
        if (i == t || !f) return f;
        while (ptr[i] < len[i]){
            int id = adj[i][ptr[i]];
            if (dis[E[id].v] == dis[i] + 1){
                long long x = dfs(E[id].v, min(f, E[id].cap - 
E[id].flow));
                if (x){
                    E[id].flow += x, E[id ^ 1].flow -= x;
                    return x;
                }
            }
            ptr[i]++;
        }
        return 0;
    }

    long long dinic(){
        long long res = 0;

        while (bfs()){
```

```
            memset(ptr, 0, n * sizeof(ptr[0]));
            while (long long f = dfs(s, INF)) {
                res += f;
            }
        }
        return res;
    }
}

namespace nodeflow{
    void init(int n, int s, int t, vector <long long> capacity){
        flow::init(n * 2, s * 2, t * 2 + 1);

        for (int i = 0; i < n; i++){
            flow::addEdge(i * 2, i * 2 + 1, capacity[i]);
        }
    }

    void addEdge(int a, int b, long long c){
        flow::addEdge(a * 2 + 1, b * 2, c);
    }

    long long dinic(){
        return flow::dinic();
    }
}

int main(){
    int n, i, j, k, a, b, c, s, t, m;

    n = MAXN - 10;
    s = 0, t = n - 1;
    flow::init(n, s, t);

    ///m = n * 1.75;
    m = 1e6;

    while (m--){
        a = ran(0, n - 1);
        b = ran(0, n - 1);
        c = ran(1, 1000000000);
        flow::addEdge(a, b, c);
    }

    clock_t start = clock();
    printf("%lld\n", flow::dinic());
    fprintf(stderr, "%0.6f\n", (clock() - start) / (1.0 *
CLOCKS_PER_SEC));
    return 0;
}
```

Dinic_s Algorithm (Matrix).cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>
```

```
#define MAXN 5010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Dinic's algorithm for directed graphs (0 based index for graphs)
/// For undirected graphs, just add two directed edges

const long long INF = (~0ULL) >> 1;

namespace flow{
   int n, s, t, ptr[MAXN], dis[MAXN], Q[MAXN];
   long long cap[MAXN][MAXN], flow[MAXN][MAXN];

   inline void init(int nodes, int source, int sink){
       clr(cap), clr(flow);
       n = nodes, s = source, t = sink;
   }

   inline void addEdge(int a, int b, int c){
       cap[a][b] += c;
   }

   inline bool bfs(){
       int i, j, f = 0, l = 0;
       memset(dis, -1, sizeof(dis[0]) * n);

       dis[s] = 0, Q[l++] = s;
       while (f < l){
           i = Q[f++];
           for (j = 0; j < n; j++){
               if (dis[j] == -1 && flow[i][j] < cap[i][j]){
                   Q[l++] = j;
                   dis[j] = dis[i] + 1;
               }
           }
       }
       return (dis[t] != -1);
   }

   long long dfs(int i, long long f){
       if (i == t || !f) return f;
       for (int &j = ptr[i]; j < n; j++){
           if (dis[j] == (dis[i] + 1)){
               long long x = dfs(j, min(f, cap[i][j] - flow[i][j]));
               if (x){
                   flow[i][j] += x, flow[j][i] -= x;
                   return x;
               }
           }
```

```
        }
        return 0;
    }

    long long dinic(){
        long long res = 0;
        while (bfs()){
            memset(ptr, 0, n * sizeof(ptr[0]));
            while (long long f = dfs(s, INF)){
                res += f;
            }
        }
        return res;
    }
}

int main(){
    int n, i, j, k, a, b, c, s, t, m;

    n = MAXN - 10;
    s = 0, t = n - 1;
    flow::init(n, s, t);

    m = n * 1.75;
    while (m--){
        a = ran(0, n - 1);
        b = ran(0, n - 1);
        c = ran(1, 1000000000);
        flow::addEdge(a, b, c);
    }

    clock_t start = clock();
    printf("%lld\n", flow::dinic());
    fprintf(stderr, "%0.6f\n", (clock() - start) / (1.0 *
CLOCKS_PER_SEC));
    return 0;
}
```

Directed MST.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

struct Edge{
    int u, v, w;

    Edge(){}
    Edge(int a, int b, int c){
```

```
        u = a, v = b, w = c;
    }
};

/// Directed minimum spanning tree in O(n * m)
/// Constructs a rooted tree of minimum total weight from the root node
/// Returns -1 if no solution from root

int directed_MST(int n, vector <Edge> E, int root){ /// hash = 547539
    const int INF = (1 << 30) - 30;

    int i, j, k, l, x, y, res = 0;
    vector <int> cost(n), parent(n), label(n), comp(n);

    for (; ;){
        for (i = 0; i < n; i++) cost[i] = INF;
        for (auto e: E){
            if (e.u != e.v && cost[e.v] > e.w){
                cost[e.v] = e.w;
                parent[e.v] = e.u;
            }
        }

        cost[root] = 0;
        for (i = 0; i < n && cost[i] != INF; i++){};
        if (i != n) return -1; /// No solution

        for (i = 0, k = 0; i < n; i++) res += cost[i];
        for (i = 0; i < n; i++) label[i] = comp[i] = -1;

        for (i = 0; i < n; i++){
            for (x = i; x != root && comp[x] == -1; x = parent[x]) comp[x]
= i;
            if (x != root && comp[x] == i){
                for (k++; label[x] == -1; x = parent[x]) label[x] = k - 1;
            }
        }

        if (k == 0) break;
        for (i = 0; i < n; i++){
            if (label[i] == -1) label[i] = k++;
        }

        for (auto &e: E){
            x = label[e.u], y = label[e.v];
            if (x != y) e.w -= cost[e.v];
            e.u = x, e.v = y;
        }
        root = label[root], n = k;
    }
    return res;
}

int main(){
```

```
    int T = 0, t, n, m, i, j, k, u, v, w, res;

    scanf("%d", &t);
    while (t--){
        vector <Edge> E;
        scanf("%d %d", &n, &m);
        for (i = 0; i < m; i++){
            scanf("%d %d %d", &u, &v, &w);
            E.push_back(Edge(u, v, w));
        }

        res = directed_MST(n, E, 0);
        if (res == -1) printf("Case #%d: Possums!\n", ++T);
        else printf("Case #%d: %d\n", ++T, res);
    }
    return 0;
}
```

Discrete Log (General).cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

int extended_gcd(int a, int b, int& x, int& y){
    /// Bezout's identity, ax + by = gcd(a,b)

    if (!b){
        y = 0, x = 1;
        return a;
    }

    int g = extended_gcd(b, a % b, y, x);
    y -= ((a / b) * x);
    return g;
}

int discrete_log(int g, int h, int p){ /// hash = 167626
    /***
        * returns smallest x such that (g^x) % p = h, -1 if none exists
        * function returns x, the discrete log of h with respect to g
modulo p
    ***/

    if (h >= p) return -1;
    if ((g % p) == 0){
        if (h == 1) return 0; /// return -1 if strictly positive integer
solution is required
        else return -1;
    }
```

```
    int i, c, x, y, z, r, m, counter = 0;
    long long v = 1, d = 1, mul = 1, temp = 1 % p;

    for (int i = 0; i < 100; i++){
        if (temp == h) return i;
        temp = (temp * g) % p;
    }

    while ((v = __gcd(g, p)) > 1){
        if (h % v) return -1;
        h /= v, p /= v;
        d = (d * (g / v)) % p;
        counter++;
    }

    m = ceil(sqrt(p)); /// may be change to sqrtl() ?
    tr1::unordered_map <int, int> mp;

    for (i = 0; i < m; i++){
        if (!mp[mul]) mp[mul] = i + 1;
        mul = (mul * g) % p;
    }

    for (i = 0; i < m; i++){
        z = extended_gcd(d, p, x, y);
        c = p / z;
        r = ((((long long)x * h) / z) % p + p) % p;
        if (mp[r]) return ((i * m) + mp[r] + counter - 1);
        d = (d * mul) % p;
    }

    return -1;
}

int main(){
    int g, h, p, res;

    for (; ;){
        scanf("%d %d %d", &g, &p, &h);
        if (!g && !p && !h) break;

        res = discrete_log(g, h % p, p);
        if (res == -1) puts("No Solution");
        else printf("%d\n", res);
    }
    return 0;
}
```

Discrete Log.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
```

```
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

int expo(long long x, int n, int m){
   long long res = 1;

   while (n){
       if (n & 1) res = (res * x) % m;
       x = (x * x) % m;
       n >>= 1;
   }

   return (res % m);
}

int extended_gcd(int a, int b, int& x, int& y){
   /// Bezout's identity, ax + by = gcd(a,b)

   if (!b){
       y = 0, x = 1;
       return a;
   }

   int g = extended_gcd(b, a % b, y, x);
   y -= ((a / b) * x);
   return g;
}

int inverse_modulo(int a, int m){
   /// inverse exists if and only if a and m are co-prime

   int x, y, inv;
   extended_gcd(a, m, x, y);
   inv = (x + m) % m;
   return inv;
}

int discrete_log(int g, int h, int p){
   /***
       * returns smallest x such that (g^x) % p = h, -1 if none exists
       * p must be a PRIME
       * function returns x, the discrete log of h with respect to g
modulo p
   ***/

   if (h >= p) return -1;
   if ((g % p) == 0){
       if (h == 1) return 0; /// return -1 if strictly positive integer
solution is required
       else return -1;
   }
```

```
   unordered_map <int, int> mp;
   int i, q, r, m = ceil(sqrt(p)); /// may be change to sqrtl() ?
   long long d = 1, inv = expo(inverse_modulo(g, p), m, p);

   for (i = 0; i <= m; i++){
       if (!mp[d]) mp[d] = i + 1;
       d *= g;
       if (d >= p) d %= p;
   }

   d = h;
   for (q = 0; q <= m; q++){
       r = mp[d];
       if (r) return ((m * q) + (--r));
       d *= inv;
       if (d >= p) d %= p;
   }

   return -1;
}

int main(){
   int T = 0, t, g, h, p;

   scanf("%d", &t);
   while (t--){
       scanf("%d %d %d", &g, &h, &p);
       int x = discrete_log(g, h, p);
       printf("Case %d: %d\n", ++T, x);
   }
   return 0;
}
```

Divisors From Factorization (Iterative).c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define LEN 78777
#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define chkbit(ar, i) (((ar[(i) >> 6]) & (1 << (((i) >> 1) & 31))))
#define setbit(ar, i) (((ar[(i) >> 6]) |= (1 << (((i) >> 1) & 31))))
#define isprime(x) (( (x) && ((x)&1) && (!chkbit(ar, (x)))) || ((x) ==
2))

int p, prime[LEN];
long long div[7001];
unsigned int ar[(MAX >> 6) + 5] = {0};

int compare(const void* a, const void* b){
   long long x = (*(long long*)a);
```

```
    long long y = (*(long long*)b);
    if (x == y) return 0;
    return ((x < y) ? -1 : 1);
}

void Sieve(){
    int i, j, k;
    setbit(ar, 0), setbit(ar, 1);

    for (i = 3; (i * i) < MAX; i++, i++){
        if (!chkbit(ar, i)){
            for (j = (i * i), k = i << 1; j < MAX; j += k) setbit(ar, j);
        }
    }

    for (i = 3, prime[0] = 2, p = 1; i < MAX; i++, i++){
        if (isprime(i)) prime[p++] = i;
    }
}

int divisors(long long x){
    int i, j, l, k, c, len = 0;
    for (i = 0, div[len++] = 1; i < p; i++){
        if ((long long)prime[i] * prime[i] > x) break;
        c = 0, k = len;
        while (!(x % prime[i])) c++, x /= prime[i];
        long long y = prime[i];
        for (j = 0; j < c; j++, y *= prime[i]){
            for (l = 0; l < k; l++) div[len++] = y * div[l];
        }
    }

    for (j = 0, k = len; j < k && x > 1; j++) div[len++] = div[j] * x;
    qsort(div, len, sizeof(long long), compare);
    return len;
}


int main(){
    Sieve();
    int i, j, k, x = divisors(2 * 3 * 5 * 21);
    printf("%d\n", x);
    for (i = 0; i < x; i++) printf("%lld\n", div[i]);
    return 0;
}

Divisors From Factorization.c
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define SQR 317
#define LEN 9777
```

```
#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define chkbit(ar, i) (((ar[(i) >> 6]) & (1 << (((i) >> 1) & 31))))
#define setbit(ar, i) (((ar[(i) >> 6]) |= (1 << (((i) >> 1) & 31))))
#define isprime(x) (( (x) && ((x)&1) && (!chkbit(ar, (x)))) || ((x) ==
2))

int p, prime[LEN];
int len, P[50], C[50], d, div[10010];
unsigned int ar[(MAX >> 6) + 5] = {0};

void Sieve(){
   int i, j, k;
   setbit(ar, 0), setbit(ar, 1);

   for (i = 3; i < SQR; i++, i++){
       if (!chkbit(ar, i)){
           k = i << 1;
           for (j = (i * i); j < MAX; j += k) setbit(ar, j);
       }
   }

   p = 0;
   prime[p++] = 2;
   for (i = 3; i < MAX; i++, i++){
       if (isprime(i)) prime[p++] = i;
   }
}

int Factorize(int* P, int* C, int n){
   int i, j, k, x, c;

   len = 0;
   for (i = 0; i < p; i++){
       x = prime[i];
       if ((x * x) > n) break;

       c = 0;
       while (!(n % x)){
           n /= x;
           c++;
       }

       if (c){
           P[len] = x;
           C[len++] = c;
           c = 0;
       }
   }
   if (n > 1){
       P[len] = n;
       C[len++] = 1;
   }
```

```
    return len;
}

void backtrack(int i, int j, int x){
    if (i == len){
        div[d++] = x;
        return;
    }

    backtrack(i + 1, 0, x);
    if (j < C[i]) backtrack(i, j + 1, x * P[i]);
}

int Divisors(int n){
    d = 0;
    Factorize(P, C, n);
    backtrack(0, 0, 1);
    return d;
}

int main(){
    Sieve();
    return 0;
}
```

Divisors From Factorization.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 10000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

namespace dv{
    long long P[100];
    bitset <MAX> isprime;
    int p = 0, C[100], primes[666666];

    void Sieve(){
        int i, j, k;
        for (i = 3; (i * i) < MAX; i += 2){
            if (!isprime[i]){
                for (j = (i * i), k = i << 1; j < MAX; j += k){
                    isprime[j] = true;
                }
            }
        }

        primes[p++] = 2;
        for (i = 3; i < MAX; i += 2){
```

```
            if (!isprime[i]) primes[p++] = i;
        }
    }

    void backtrack(int i, long long d, vector <long long>& v){
        if (i < 0){
            v.push_back(d);
            return;
        }
        for (int j = 0; j <= C[i]; j++, d *= P[i]) backtrack(i - 1, d, v);
    }

    vector <long long> divisors(long long n){
        if (!p) Sieve();

        int i, k, x, len = 0;
        for (i = 0; i < p; i++){
            k = 0, x = primes[i];
            if (((long long)x * x) > n) break;

            while (!(n % x)){
                k++;
                n /= x;
            }
            if (k) P[len] = x, C[len++] = k;
        }
        if (n > 1) P[len] = n, C[len++] = 1;

        vector <long long> v;
        backtrack(len - 1, 1, v);
        sort(v.begin(), v.end());
        return v;
    }
}

int main(){

}
```

Divisors in Range.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define SQR 10001
#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

short P[MAX];
int L[MAX], ar[MAX];

void Generate(){
```

```
    int i, j, d, x, y;

    P[0] = P[1] = L[0] = L[1] = 1;
    for (i = 4; i < MAX; i++, i++) P[i] = 2;

    for (i = 3; i < SQR; i++, i++){
        if (!P[i]){
            d = i << 1;
            for (j = (i * i); j < MAX; j += d) P[j] = i;
        }
    }

    for (i = 2; i < MAX; i++){
        if (!P[i]) L[i] = i;
        else{
            L[i] = P[i];
            x = L[i /P[i]];
            if (x > L[i]) L[i] = x;
        }
    }

    ar[0] = 0, ar[1] = 1;
    for (i = 2; i < MAX; i++){
        if (L[i] == i) ar[i] = 2;
        else{
            x = i, y = 1;
            while (L[x] == L[i]){
                y++;
                x /= L[i];
            }
            ar[i] = (ar[x] * y);
        }
    }
}

int main(){
    Generate();
    int t, n, i, r;
    long long sum = 0;
    return 0;
}
```

Double Hashing.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

struct simplehash{
    int len;
```

```
   long long base, mod;
   vector <int> P, H, R;

   simplehash(){}
   simplehash(const char* str, long long b, long long m){
       base = b, mod = m, len = strlen(str);
       P.resize(len + 3, 1), H.resize(len + 3, 0), R.resize(len + 3, 0);

       for (int i = 1; i <= len; i++) P[i] = (P[i - 1] * base) % mod;
       for (int i = 1; i <= len; i++) H[i] = (H[i - 1] * base + str[i -
1] + 1007) % mod;
       for (int i = len; i >= 1; i--) R[i] = (R[i + 1] * base + str[i -
1] + 1007) % mod;
   }

   inline int range_hash(int l, int r){
       int hashval = H[r + 1] - ((long long)P[r - l + 1] * H[l] % mod);
       return (hashval < 0 ? hashval + mod : hashval);
   }

   inline int reverse_hash(int l, int r){;
       int hashval = R[l + 1] - ((long long)P[r - l + 1] * R[r + 2] %
mod);
       return (hashval < 0 ? hashval + mod : hashval);
   }
};

struct stringhash{
   simplehash sh1, sh2;
   stringhash(){}
   stringhash(const char* str){
       sh1 = simplehash(str, 1949313259, 2091573227);
       sh2 = simplehash(str, 1997293877, 2117566807);
   }

   inline long long range_hash(int l, int r){
       return ((long long)sh1.range_hash(l, r) << 32) ^ sh2.range_hash(l,
r);
   }

   inline long long reverse_hash(int l, int r){
       return ((long long)sh1.reverse_hash(l, r) << 32) ^
sh2.reverse_hash(l, r);
   }
};

int main(){

}
```

Eulerian Numbers.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
```

```
#include <stdbool.h>
#include <assert.h>

#define MAX 5010
#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

/***

Eulerian number A(n,k) is the number of permutations of 1 to n in which
exactly k elements are greater than their previous element
Eulerian triangle for n = 1 to 7 and k = 0 to n - 1 below


1
1 1
1 4 1
1 11 11 1
1 26 66 26 1
1 57 302 302 57 1
1 120 1191 2416 1191 120 1

***/

int dp[MAX][MAX];

void generate(){
   int n, k;
   for (n = 0; n < MAX; n++){
       for (k = 1, dp[n][0] = 1; k <= n; k++){
           dp[n][k] = ((long long)dp[n - 1][k] * (k + 1) + (long
long)dp[n - 1][k - 1] * (n - k)) % MOD;
       }
   }
}

int main(){
   int n, k;
   generate();

   for (n = 1; n <= 7; n++){
       for (k = 0; k < n; k++){
           printf("%d ", dp[n][k]);
       }
       puts("");
   }
   return 0;
}

Extended Euclid.cpp
--------------------------------------------------
#include <bits/stdtr1c++.h>
```

```
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Bezout's identity, ax + by = gcd(a,b)

int extended_gcd(int a, int b, int& x, int& y){

   if (!b){
       y = 0, x = 1;
       return a;
   }

   int g = extended_gcd(b, a % b, y, x);
   y -= ((a / b) * x);
   return g;
}

/// Linear diophantine equation, ax + by = c

void diophantine(int a, int b, int c, int& x, int& y){
   int g = extended_gcd(a, b, x, y);
   assert((c % g) == 0); /// c must be a multiply of g

   c /= g;
   x *= c, y *= c;
}

int main(){
}
```

FFT 2.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

typedef complex<double> complx;

/// Computes the discrete fourier transformation of the vector when inv =
1
/// Computes the inverse discrete fourier transformation when inv = -1

vector <complx> FFT(vector <complx> ar, int inv){
   int i, j, l, len, n = ar.size();
   const double p = 4.0 * inv * acos(0.0);
```

```
    for (i = 1, j = 0; i < n; i++){
        for (l = n >> 1; j >= l; l >>= 1) j -= l;
        j += l;
        if (i < j) swap(ar[i], ar[j]);
    }

    for (l = 1, len = 2; len <= n; l <<= 1, len <<= 1){
        double theta = p / len;
        complx mul(cos(theta), sin(theta));

        for (i = 0; i < n; i += len){
            complx w(1.0, 0.0);
            for (j = 0; j < l; j++){
                complx u = ar[i + j], v = ar[i + j + l] * w;
                ar[i + j] = u + v, ar[i + j + l] = u - v;
                w *= mul;
            }
        }
    }

    if (inv == -1){
        for (i = 0; i < n; i++) ar[i] /= n;
    }
    return ar;
}

/***
    Computes the circular convolution of A and B, denoted A * B
    A and B must be of equal size, if not normalize before calling
function
    Example to demonstrate convolution for n = 5:

    c0 = a0b0 + a1b4 + a2b3 + a3b2 + a4b1
    c1 = a0b1 + a1b0 + a2b4 + a3b3 + a4b2
    ...
    c4 = a0b4 + a1b3 + a2b2 + a3b1 + a4b0


    Note: If linear convolution is required, pad with zeros appropriately,
as in multiplication

***/

vector <complx> convolution(vector <complx> A, vector <complx> B){
    int n, m, i;
    n = A.size();
    vector <complx> C;
    m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) == 1));
    A.resize(m, 0), B.resize(m, 0), C.resize(m, 0);

    A = FFT(A, 1), B = FFT(B, 1);
    for (i = 0; i < m; i++) C[i] = A[i] * B[i];
    return FFT(C, -1);
}
```

```
/// Multiplies two polynomials A and B and return the coefficients of
their product
vector <complx> multiply(vector <complx> A, vector <complx> B){
   int n, m, i;
   vector <complx> C;
   n = A.size() + B.size() - 1;
   m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) == 1));
   A.resize(m, 0), B.resize(m, 0), C.resize(m, 0);

   A = FFT(A, 1), B = FFT(B, 1);
   for (i = 0; i < m; i++) C[i] = A[i] * B[i];
   return FFT(C, -1);
}

int main(){
   vector <complx> A = {1, 2, 3, 4};
   vector <complx> B = {7, 29, 13, 17};


   vector <complx> C = convolution(A, B);
   for (int i = 0; i < C.size(); i++) dbg(C[i].real());
}
```

FFT Extended + OP.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAXN 1048576 /// 2 * MAX at least
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Change long double to double if not required

namespace fft{
   int len, last = -1, step = 0, rev[MAXN];
   long long C[MAXN], D[MAXN], P[MAXN], Q[MAXN];

      struct complx{
       long double real, img;

       inline complx(){
           real = img = 0.0;
       }

       inline complx conjugate(){
           return complx(real, -img);
       }

       inline complx(long double x){
```

```
            real = x, img = 0.0;
        }

        inline complx(long double x, long double y){
            real = x, img = y;
        }

        inline complx operator + (complx other){
            return complx(real + other.real, img + other.img);
        }

        inline complx operator - (complx other){
            return complx(real - other.real, img - other.img);
        }

        inline complx operator * (complx other){
            return complx((real * other.real) - (img * other.img), (real *
other.img) + (img * other.real));
        }
    } u[MAXN], v[MAXN], f[MAXN], g[MAXN], dp[MAXN], inv[MAXN];

    /// Pre-process roots, inverse roots and fft leaf index
    void build(int& a, long long* A, int& b, long long* B){
        while (a > 1 && A[a - 1] == 0) a--;
        while (b > 1 && B[b - 1] == 0) b--;

           len = 1 << (32 - __builtin_clz(a + b) - (__builtin_popcount(a +
b) == 1));
        for (int i = a; i < len; i++) A[i] = 0;
        for (int i = b; i < len; i++) B[i] = 0;

             if (!step++){
            dp[1] = inv[1] = complx(1);
            for (int i = 1; (1 << i) < MAXN; i++){
                double theta = (2.0 * acos(0.0)) / (1 << i);
                complx mul = complx(cos(theta), sin(theta));
                complx inv_mul = complx(cos(-theta), sin(-theta));

                int lim = 1 << i;
                for (int j = lim >> 1; j < lim; j++){
                    dp[2 * j] = dp[j], inv[2 * j] = inv[j];
                    inv[2 * j + 1] = inv[j] * inv_mul;
                    dp[2 * j + 1] = dp[j] * mul;
                }
            }
        }

        if (last != len){
            last = len;
            int bit = (32 - __builtin_clz(len) - (__builtin_popcount(len)
== 1));
            for (int i = 0; i < len; i++) rev[i] = (rev[i >> 1] >> 1) +
((i & 1) << (bit - 1));
        }
```

```
    }

  /// Fast Fourier Transformation, iterative divide and conquer
    void transform(complx *in, complx *out, complx* ar){
        for (int i = 0; i < len; i++) out[i] = in[rev[i]];
        for (int k = 1; k < len; k <<= 1){
         for (int i = 0; i < len; i += (k << 1)){
             for (int j = 0; j < k; j++){
                 complx z = out[i + j + k] * ar[j + k];
                 out[i + j + k] = out[i + j] - z;
                 out[i + j] = out[i + j] + z;
             }
         }
        }
    }

  /// Fast Fourier Transformation, iterative divide and conquer unrolled
and optimized
    void transform_unrolled(complx *in, complx *out, complx* ar){
        for (int i = 0; i < len; i++) out[i] = in[rev[i]];
        for (int k = 1; k < len; k <<= 1){
         for (int i = 0; i < len; i += (k << 1)){
             complx z, *a = out + i, *b = out + i + k, *c = ar + k;
             if (k == 1){
                 z = (*b) * (*c);
                 *b = *a - z, *a = *a + z;
             }

             for (int j = 0; j < k && k > 1; j += 2, a++, b++, c++){
                 z = (*b) * (*c);
                 *b = *a - z, *a = *a + z;
                 a++, b++, c++;
                 z = (*b) * (*c);
                 *b = *a - z, *a = *a + z;
             }
         }
        }
    }

    bool equals(int a, long long* A, int b, long long* B){
     if (a != b) return false;
     for (a = 0; a < b && A[a] == B[a]; a++){}
     return (a == b);
    }

  /// Square of a polynomial
    int square(int a, long long* A){
     build(a, A, a, A);
     for (int i = 0; i < len; i++) u[i] = complx(A[i], 0);
     transform_unrolled(u, f, dp);
     for (int i = 0; i < len; i++) u[i] = f[i] * f[i];
     transform_unrolled(u, f, inv);
     for (int i = 0; i < len; i++) A[i] = (f[i].real / (long
double)len) + 0.5;
```

```
        return a + a - 1;
       }

   /// Multiplies two polynomials A and B and return the coefficients of
their product in A
   /// Function returns degree of the polynomial A * B
      int multiply(int a, long long* A, int b, long long* B){
          if (equals(a, A, b, B)) return square(a, A); /// Optimization

            build(a, A, b, B);
            for (int i = 0; i < len; i++) u[i] = complx(A[i], B[i]);
            transform_unrolled(u, f, dp);
            for (int i = 0; i < len; i++){
            int j = (len - 1) & (len - i);
            u[i] = (f[j] * f[j] - f[i].conjugate() * f[i].conjugate()) *
complx(0, -0.25 / len);
            }
            transform_unrolled(u, f, dp);
            for (int i = 0; i < len; i++) A[i] = f[i].real + 0.5;
            return a + b - 1;
      }

   /// Modular multiplication
      int mod_multiply(int a, long long* A, int b, long long* B, int
mod){
          build(a, A, b, B);
          int flag = equals(a, A, b, B);
          for (int i = 0; i < len; i++) A[i] %= mod, B[i] %= mod;
          for (int i = 0; i < len; i++) u[i] = complx(A[i] & 32767, A[i]
>> 15);
          for (int i = 0; i < len; i++) v[i] = complx(B[i] & 32767, B[i]
>> 15);

            transform_unrolled(u, f, dp);
            for (int i = 0; i < len; i++) g[i] = f[i];
            if (!flag) transform_unrolled(v, g, dp);

            for (int i = 0; i < len; i++){
            int j = (len - 1) & (len - i);
            complx c1 = f[j].conjugate(), c2 = g[j].conjugate();

            complx a1 = (f[i] + c1) * complx(0.5, 0);
                  complx a2 = (f[i] - c1) * complx(0, -0.5);
                  complx b1 = (g[i] + c2) * complx(0.5 / len, 0);
                  complx b2 = (g[i] - c2) * complx(0, -0.5 / len);
                  v[j] = a1 * b2 + a2 * b1;
                  u[j] = a1 * b1 + a2 * b2 * complx(0, 1);
            }
            transform_unrolled(u, f, dp);
            transform_unrolled(v, g, dp);

        long long x, y, z;
            for (int i = 0; i < len; i++){
            x = f[i].real + 0.5, y = g[i].real + 0.5, z = f[i].img + 0.5;
```

```
            A[i] = (x + ((y % mod) << 15) + ((z % mod) << 30)) % mod;
            }
            return a + b - 1;
      }

   /// Multiplies two polynomials where intermediate and final values
fits in long long
      int long_multiply(int a, long long* A, int b, long long* B){
            int mod1 = 1.5e9;
            int mod2 = mod1 + 1;
            for (int i = 0; i < a; i++) C[i] = A[i];
            for (int i = 0; i < b; i++) D[i] = B[i];

            mod_multiply(a, A, b, B, mod1);
            mod_multiply(a, C, b, D, mod2);
            for (int i = 0; i < len; i++){
            A[i] = A[i] + (C[i] - A[i] + (long long)mod2) * (long
long)mod1 % mod2 * mod1;
            }
            return a + b - 1;
      }

      int build_convolution(int n, long long* A, long long* B){
       int i, m, d = 0;
       for (i = 0; i < n; i++) Q[i] = Q[i + n] = B[i];
       for (i = 0; i < n; i++) P[i] = A[i], P[i + n] = 0;
       n *= 2, m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n)
== 1));
       for (i = n; i < m; i++) P[i] = Q[i] = 0;
       return n;
      }

      /***
       Computes the circular convolution of A and B, denoted A * B, in C
       A and B must be of equal size, if not normalize before calling
function
       Example to demonstrate convolution for n = 5:

       c0 = a0b0 + a1b4 + a2b3 + a3b2 + a4b1
       c1 = a0b1 + a1b0 + a2b4 + a3b3 + a4b2
       ...
       c4 = a0b4 + a1b3 + a2b2 + a3b1 + a4b0


       Note: If linear convolution is required, pad with zeros
appropriately, as in multiplication

   ***/

   /// Returns the convolution of A and B in A
      void convolution(int n, long long* A, long long* B){
       int len = build_convolution(n, A, B);
       multiply(len, P, len, Q);
       for (int i = 0; i < n; i++) A[i] = P[i + n];
```

```
    }

  /// Modular convolution
    void mod_convolution(int n, long long* A, long long* B, int mod){
     int len = build_convolution(n, A, B);
     mod_multiply(len, P, len, Q, mod);
     for (int i = 0; i < n; i++) A[i] = P[i + n];
    }

  /// Convolution in long long
    void long_convolution(int n, long long* A, long long* B){
     int len = build_convolution(n, A, B);
     long_multiply(len, P, len, Q);
     for (int i = 0; i < n; i++) A[i] = P[i + n];
    }

    /// Hamming distance vector of B with every substring of length
|pattern| in str
  /// str and pattern consists of only '1' and '0'
  /// str = "01111000010011111111110010001101000100011110101111"
  /// pattern = "1001101001101110101101000"
  /// Sum of values in hamming distance vector = 321

    vector <int> hamming_distance(const char* str, const char*
pattern){
     int n = strlen(str), m = strlen(pattern);
     for (int i = 0; i < n; i++) P[i] = Q[i] = 0;
     for (int i = 0; i < n; i++) P[i] = str[i] == '1' ? 1 : -1;
     for (int i = 0, j = m - 1; j >= 0; i++, j--) Q[i] = pattern[j] ==
'1' ? 1 : -1;

     vector <int> res;
     fft::multiply(n, P, m, Q);
     for (int i = 0; (i + m) <= n; i++){
         res.push_back(m - ((P[i + m - 1] + m) >> 1));
     }
     return res;
    }
}

int main(){

}
```

FFT Extended.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 262148 /// 2 * Smallest power of 2 greater than MAXN, 2^18
when MAXN = 10^5
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))
```

```
using namespace std;

/// Replace long double with double if not required

namespace fft{
   struct complx{ /// hash = 463718 (including P, P1, P2 arrays)
        long double real, img;

        inline complx(){
            real = img = 0.0;
        }

        inline complx(long double x){
            real = x, img = 0.0;
        }

        inline complx(long double x, long double y){
            real = x, img = y;
        }

        inline void operator += (complx &other){
            real += other.real, img += other.img;
        }

        inline void operator -= (complx &other){
            real -= other.real, img -= other.img;
        }

        inline complx operator + (complx &other){
            return complx(real + other.real, img + other.img);
        }

        inline complx operator - (complx &other){
            return complx(real - other.real, img - other.img);
        }

        inline complx operator * (complx& other){
            return complx((real * other.real) - (img * other.img), (real *
other.img) + (img * other.real));
        }
   } P[MAX >> 1], P1[MAX], P2[MAX];

   int rev[MAX];
   long long ar[6][MAX];

   void transform(complx *ar, int n, int inv){ /// hash = 131109
        int i, j, k, l, len, len2;
        const long double p = 4.0 * inv * acos(0.0);

        for (i = 0, l = __builtin_ctz(n) - 1; i < n; i++){
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << l);
            if(i < rev[i]) swap(ar[i], ar[rev[i]]);
        }
```

```
        for (len = 2; len <= n; len <<= 1){
            len2 = len >> 1;
            long double theta = p / len;
            complx mul(cos(theta), sin(theta));
            for (i = 1, P[0] = complx(1, 0); i < len2; i++) P[i] = (P[i -
1] * mul);

            for (i = 0; i < n; i += len){
                complx t, *x = ar + i, *y = ar + i + len2, *l = ar + i +
len2, *z = P;
                for (; x != l; x++, y++, z++){
                    t = (*y) * (*z), *y = *x - t;
                    *x += t;
                }
            }
        }

        if (inv == -1){
            long double tmp = 1.0 / n;
            for (i = 0; i < n; i++) ar[i].real *= tmp;
        }
    }

    void fft_convolution(int n, complx* A, complx* B){ /// hash = 121635
        int i, m, d = 0;
        if (__builtin_popcount(n) != 1){
            for (i = 0; i < n; i++) B[i + n] = B[i], A[i + n] = complx(0);
            d = n, n *= 2;
        }

        m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) == 1));
        for (i = n; i < m; i++) A[i] = B[i] = complx(0);

        transform(A, m, 1), transform(B, m, 1);
        for (i = 0; i < m; i++) A[i] = A[i] * B[i];
        transform(A, m, -1);
        for (i = 0; i < d && d; i++) A[i] = A[i + d];
    }

    int fft_multiply(int a, complx* A, int b, complx* B){
        int n = a + b - 1;
        int m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) ==
1));
        for (int i = a; i < m; i++) A[i] = complx(0);
        for (int i = b; i < m; i++) B[i] = complx(0);
        transform(A, m, 1), transform(B, m, 1);
        for (int i = 0; i < m; i++) A[i] = A[i] * B[i];
        transform(A, m, -1);
        return m;
    }

    /// Multiplies two polynomials A and B and return the coefficients of
their product in A
```

```
/// Function returns degree of the polynomial A * B

template <typename dataType>
int multiply(int a, dataType* A, int b, dataType* B){
    for (int i = 0; i < a; i++) P1[i] = complx(A[i], 0.0);
    for (int i = 0; i < b; i++) P2[i] = complx(B[i], 0.0);
    int len = fft_multiply(a, P1, b, P2);
    for (int i = 0; i < len; i++) A[i] = floor(P1[i].real + 0.5);
    return len;
}

/***
    Computes the circular convolution of A and B, denoted A * B, in C
    A and B must be of equal size, if not normalize before calling
function
    Example to demonstrate convolution for n = 5:

    c0 = a0b0 + a1b4 + a2b3 + a3b2 + a4b1
    c1 = a0b1 + a1b0 + a2b4 + a3b3 + a4b2
    ...
    c4 = a0b4 + a1b3 + a2b2 + a3b1 + a4b0


    Note: If linear convolution is required, pad with zeros
appropriately, as in multiplication

***/

/// Returns the convolution of A and B in A
template <typename dataType>
void convolution(int n, dataType* A, dataType* B){
    for (int i = 0; i < n; i++) P1[i] = complx(A[i], 0.0), P2[i] =
complx(B[i], 0.0);
    fft_convolution(n, P1, P2);
    for (int i = 0; i < n; i++) A[i] = floor(P1[i].real + 0.5);
}

/// Extended functions

/// Returns the convolution of A and B modulo mod in A
void convolution(int n, int* A, int* B, int mod){ /// hash = 174371
    int i, s = sqrt(0.9 + mod);

    for (i = 0; i < n; i++){
        ar[0][i] = ar[5][i] = A[i] % s;
        ar[1][i] = ar[4][i] = A[i] / s;
    }
    for (i = 0; i < n; i++) ar[2][i] = B[i] % s, ar[3][i] = B[i] / s;

    convolution(n, ar[5], ar[2]), convolution(n, ar[0], ar[3]);
    convolution(n, ar[4], ar[2]), convolution(n, ar[1], ar[3]);

    for (i = 0; i < n; i++){
```

```
            A[i] = ( ((ar[0][i] + ar[4][i]) * s) + ar[5][i] + (ar[1][i] *
s % mod * s)) % mod;
        }
    }

    /// Multiplies two polynomials A and B and return the coefficients of
their product in A modulo m
    /// Function returns degree of the polynomial A * B

    int multiply(int a, int* A, int b, int* B, int mod){ /// hash = 848260
        int i, s = sqrt(0.9 + mod);

        for (i = 0; i < a; i++){
            ar[0][i] = ar[5][i] = A[i] % s;
            ar[1][i] = ar[4][i] = A[i] / s;
        }
        for (i = 0; i < b; i++) ar[2][i] = B[i] % s, ar[3][i] = B[i] / s;

        multiply(a, ar[5], b, ar[2]), multiply(a, ar[0], b, ar[3]);
        multiply(a, ar[4], b, ar[2]), multiply(a, ar[1], b, ar[3]);

        for (i = 0; i < (a + b - 1); i++){
            A[i] = ( ((ar[0][i] + ar[4][i]) * s) + ar[5][i] + (ar[1][i] *
s % mod * s)) % mod;
        }
        return (a + b - 1);
    }

    /// Multiplies two polynomials A and B and return the coefficients of
their product in A modulo m
    /// Function returns degree of the polynomial A * B

    int fastmul(int a, int* A, int b, int* B, int mod){
        complx ar[4][MAX]; /// make global if RTE
        int i, m, n = a + b - 1, s = sqrt(0.9 + mod);

        m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) == 1));
        for (i = 0; i < a; i++) ar[0][i] = complx(A[i] % s, 0), ar[1][i] =
complx(A[i] / s, 0);
        for (i = 0; i < b; i++) ar[2][i] = complx(B[i] % s, 0), ar[3][i] =
complx(B[i] / s, 0);
        for (i = a; i < m; i++) ar[0][i] = ar[1][i] = complx(0);
        for (i = b; i < m; i++) ar[2][i] = ar[3][i] = complx(0);

        for (i = 0; i < 4; i++) transform(ar[i], m, 1);
        for (i = 0; i < m; i++){
            P1[i] = ar[0][i] * ar[2][i], P2[i] = ar[1][i] * ar[2][i];
            ar[0][i] = ar[0][i] * ar[3][i], ar[1][i] = ar[1][i] *
ar[3][i];
        }
        transform(ar[0], m, -1), transform(ar[1], m, -1), transform(P2, m,
-1), transform(P1, m, -1);

        for (i = 0; i < (a + b - 1); i++){
```

```
            long long x = floor(ar[0][i].real + 0.5), y = floor(P2[i].real
+ 0.5);
            long long z = floor(P1[i].real + 0.5), w = floor(ar[1][i].real
+ 0.5);
            A[i] = ( ((x + y) * s) + z + (w * s % mod * s)) % mod;
        }
        return (a + b - 1);
    }
}

int main(){

}
```

FFT OP 2.cpp

```
#include <bits/stdtr1c++.h>

#define TMAX 10010
#define CUTOFF 512
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

#pragma OPT_LEVEL 4
#pragma GCC OPTIMIZE("+Oaggressive")
#pragma GCC OPTIMIZE("+Onolimit")
#pragma GCC OPTIMIZE("+Oall")

using namespace std;

struct complx{
    double real, img;

    inline complx(){
        real = img = 0.0;
    }

    inline complx(double x){
        real = x, img = 0.0;
    }

    inline complx(double x, double y){
        real = x, img = y;
    }

    inline void operator += (complx &other){
        real += other.real, img += other.img;
    }

    inline void operator -= (complx &other){
        real -= other.real, img -= other.img;
    }
```

```
    inline complx operator + (complx &other){
        return complx(real + other.real, img + other.img);
    }

    inline complx operator - (complx &other){
        return complx(real - other.real, img - other.img);
    }

    inline complx operator * (complx& other){
        return complx((real * other.real) - (img * other.img), (real *
other.img) + (img * other.real));
    }
};

complx wlen_P[TMAX >> 1], A[TMAX], B[TMAX], C[TMAX];

void FFT(complx *ar, int n, int inv){
    int i, j, l, len, len2;
    const double p = 4.0 * inv * acos(0.0);

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

    for (len = 2; len <= n; len <<= 1){
        len2 = len >> 1;
        double theta = p / len;
        complx mul(cos(theta), sin(theta));

        wlen_P[0] = complx(1, 0);
        for (i = 1; i < len2; i++) wlen_P[i] = (wlen_P[i - 1] * mul);

        for (i = 0; i < n; i += len){
            complx t, *pu = ar + i, *pv = ar + i + len2, *pend = ar + i +
len2, *pw = wlen_P;
            for (; pu != pend; pu++, pv++, pw++){
                t = (*pv) * (*pw);
                *pv = *pu - t;
                *pu += t;
            }
        }
    }

    if (inv == -1){
        double tmp = 1.0 / n;
        for (i = 0; i < n; i++) ar[i].real *= tmp; /// ar[i].img is
unchanged because of optimization, add ar[i].img *= tmp if imaginary part
is also required
    }
}

void convolution(int n, complx* A, complx* B, complx* C){
```

```
   int i, m;
   m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) == 1));

   for (i = n; i < m; i++) A[i] = B[i] = complx(0);
   FFT(A, m, 1), FFT(B, m, 1);
   for (i = 0; i < m; i++) C[i] = A[i] * B[i];
   FFT(C, m, -1);
}

int multiply(int a, int b, complx* A, complx* B, complx* C){
   int i, j, n, m;

   if (((long long)a * b) < CUTOFF){
       for (i = 0; i < (a + b); i++) C[i] = complx(0);

       for (i = 0; i < a; i++){
           for (j = 0; j < b; j++){
               complx tmp = A[i] * B[j];
               C[i + j] += tmp;
           }
       }
       return (a + b - 1);
   }

   n = a + b - 1;
   m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) == 1));

   for (i = a; i < m; i++) A[i] = complx(0);
   for (i = b; i < m; i++) B[i] = complx(0);
   FFT(A, m, 1), FFT(B, m, 1);
   for (i = 0; i < m; i++) C[i] = A[i] * B[i];
   FFT(C, m, -1);

   return m;
}

int main(){
}
```

FFT OP.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define TMAX 10010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

typedef complex <double> complx;

complx wlen_P[TMAX >> 1], A[TMAX], B[TMAX], C[TMAX];
```

```
void FFT(complx *ar, int n, int inv){
   int i, j, l, len, len2;
   const double p = 4.0 * inv * acos(0.0);

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

   for (len = 2; len <= n; len <<= 1){
       len2 = len >> 1;
       double theta = p / len;
       complx mul(cos(theta), sin(theta));

       wlen_P[0] = complx(1, 0);
       for (i = 1; i < len2; i++) wlen_P[i] = (wlen_P[i - 1] * mul);

       for (i = 0; i < n; i += len){
           complx t, *pu = ar + i, *pv = ar + i + len2, *pend = ar + i +
len2, *pw = wlen_P;
           for (; pu != pend; pu++, pv++, pw++){
               t = (*pv) * (*pw);
               *pv = *pu - t;
               *pu += t;
           }
       }
   }

   if (inv == -1){
       for (i = 0; i < n; i++) ar[i] /= n;
   }
}

void convolution(int n, complx* A, complx* B, complx* C){
   int i, m;
   m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) == 1));

   for (i = n; i < m; i++) A[i] = B[i] = 0;
   FFT(A, m, 1), FFT(B, m, 1);
   for (i = 0; i < m; i++) C[i] = A[i] * B[i];
   FFT(C, m, -1);
}

int multiply(int a, int b, complx* A, complx* B, complx* C){
   int i, n, m;
   n = a + b - 1;
   m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) == 1));

   for (i = a; i < m; i++) A[i] = 0;
   for (i = b; i < m; i++) B[i] = 0;
   FFT(A, m, 1), FFT(B, m, 1);
   for (i = 0; i < m; i++) C[i] = A[i] * B[i];
   FFT(C, m, -1);
```

```
   return m;
}

int main(){
}
```

FFT.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 262148 /// 2 * Smallest power of 2 greater than MAXN, 2^18
when MAXN = 10^5
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

typedef complex <double> complx; /// Replace double with long double if
more precision is required

complx dp[MAX >> 1], P1[MAX], P2[MAX];

void FFT(complx *ar, int n, int inv){
   int i, j, l, len, len2;
   const double p = 4.0 * inv * acos(0.0);

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

   for (len = 2; len <= n; len <<= 1){
       len2 = len >> 1;
       double theta = p / len;
       complx mul(cos(theta), sin(theta));

       dp[0] = complx(1, 0);
       for (i = 1; i < len2; i++) dp[i] = (dp[i - 1] * mul);

       for (i = 0; i < n; i += len){
           complx t, *pu = ar + i, *pv = ar + i + len2, *pend = ar + i +
len2, *pw = dp;
           for (; pu != pend; pu++, pv++, pw++){
               t = (*pv) * (*pw);
               *pv = *pu - t;
               *pu += t;
           }
       }
   }
```

```
    if (inv == -1){
        for (i = 0; i < n; i++) ar[i] /= n;
    }
}

void convolution(int n, complx* A, complx* B){
    int i, m, d = 0;
    if (__builtin_popcount(n) != 1){
        for (i = 0; i < n; i++) B[i + n] = B[i], A[i + n] = complx(0);
        d = n, n *= 2;
    }

    m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) == 1));
    for (i = n; i < m; i++) A[i] = B[i] = complx(0);

    FFT(A, m, 1), FFT(B, m, 1);
    for (i = 0; i < m; i++) A[i] = A[i] * B[i];
    FFT(A, m, -1);
    for (i = 0; i < d && d; i++) A[i] = A[i + d];
}

int multiply(int a, complx* A, int b, complx* B){
    int i, n, m;
    n = a + b - 1;
    m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) == 1));

    for (i = a; i < m; i++) A[i] = 0;
    for (i = b; i < m; i++) B[i] = 0;
    FFT(A, m, 1), FFT(B, m, 1);
    for (i = 0; i < m; i++) A[i] = A[i] * B[i];
    FFT(A, m, -1);
    return m;
}

/***
    Computes the circular convolution of A and B, denoted A * B, in C
    A and B must be of equal size, if not normalize before calling
function
    Example to demonstrate convolution for n = 5:

    c0 = a0b0 + a1b4 + a2b3 + a3b2 + a4b1
    c1 = a0b1 + a1b0 + a2b4 + a3b3 + a4b2
    ...
    c4 = a0b4 + a1b3 + a2b2 + a3b1 + a4b0


    Note: If linear convolution is required, pad with zeros appropriately,
as in multiplication

***/

/// Returns the convolution of A and B in A

void convolution(int n, int* A, int* B){
```

```
    for (int i = 0; i < n; i++) P1[i] = complx(A[i], 0);
    for (int i = 0; i < n; i++) P2[i] = complx(B[i], 0);
    convolution(n, P1, P2);
    for (int i = 0; i < n; i++) A[i] = floor(P1[i].real() + 0.5);
}

/// Multiplies two polynomials A and B and return the coefficients of
their product in A
/// Function returns degree of the polynomial A * B

int multiply(int a, int* A, int b, int* B){
    for (int i = 0; i < a; i++) P1[i] = complx(A[i], 0);
    for (int i = 0; i < b; i++) P2[i] = complx(B[i], 0);
    int degree = multiply(a, P1, b, P2);
    for (int i = 0; i < degree; i++) A[i] = floor(P1[i].real() + 0.5);
    return degree;
}

int main(){
}
```

Fast Exponentiation Without Overflow.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

long long mul(long long a, long long b, long long m){
    long long res = 0;
    long long x = (a % m);

    while (b){
        if (b & 1){
            res = res + x;
            if (res >= m) res -= m;
        }
        b >>= 1;
        x <<= 1;
        if (x >= m) x -= m;
    }

    return res;
}

long long expo(long long x, long long n, long long m){
    long long res = 1;

    while (n){
        if (n & 1) res = mul(res, x, m);
        x = mul(x, x, m);
        n >>= 1;
```

```
    }

    return (res % m);
}

int main(){

}
```

Fast Exponentiation.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int expo(int a, int b){
    int res = 1;

    while (b){
        if (b & 1) res = (long long)res * a % MOD;
        a = (long long)a * a % MOD;
        b >>= 1;
    }
    return res;
}

long long expo(int x, int n, int m){
    if (!n) return (1 % m);
    else if (n & 1) return ((expo(x, n - 1, m) * x) % m);
    else{
        long long r = expo(x, n >> 1, m);
        return ((r * r) % m);
    }
}

long long expo(long long x, long long n, long long m){
    if (!n) return (1LL % m);
    else if (n & 1LL) return ((expo(x, n - 1, m) * x) % m);
    else{
        long long r = expo(x, n >> 1LL, m);
        return ((r * r) % m);
    }
}

long long expo(long long x, long long n, long long m){
    x %= m;
    long long res = 1;

    while (n){
        if (n & 1LL) res = (res * x) % m;
        x = (x * x) % m;
```

```
        n >>= 1LL;
    }

    return (res % m);
}

int expo(long long int x, int n, int m){
    x %= m;
    long long res = 1;

    while (n){
        if (n & 1) res = (res * x) % m;
        x = (x * x) % m;
        n >>= 1;
    }

    return (res % m);
}

int expo(long long x, int n){
    x %= MOD;
    long long res = 1;

    while (n){
        if (n & 1) res = (res * x) % MOD;
        x = (x * x) % MOD;
        n >>= 1;
    }

    return (res % MOD);
}

long long expo(long long x, long long n){
    x %= MOD;
    long long res = 1;

    while (n){
        if (n & 1LL) res = (res * x) % MOD;
        x = (x * x) % MOD;
        n >>= 1LL;
    }

    return (res % MOD);
}

int main(){
}
```

Fast Fibonacci LL MOD.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
```

```
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

long long n, MOD = 1000000007;

long long mul(long long a, long long b){
  if ((MOD < 3037000500LL)) return ((a * b) % MOD);
  long double res = a;
  res *= b;
  long long c = (long long)(res / MOD);
  a *= b;
  a -= c * MOD;
  if (a >= MOD) a -= MOD;
  if (a < 0) a += MOD;
  return a;
}

/*** Fast Doubling Algorithm to calculate n'th fibonacci number ***/
/*** fib(0) = 0, fib(1) = 1, fib(n) = fib(n - 1) + fib(n - 2) ***/

inline long long fib(long long& x, long long& y, long long n){
   if (!n) x = 0, y = 1;
   else{
       long long a, b;
       fib(a, b, n >> 1);
       long long z = (b << 1) - a;
       if (z < 0) z += MOD;

       x = mul(a, z);
       y = mul(a, a) + mul(b, b);
       if (y >= MOD) y -= MOD;

       if (n & 1){
           x += y;
           if (x >= MOD) x -= MOD;
           x ^= y ^= x ^= y;
       }
   }
   return x;
}

inline long long fib(long long n){
   long long x = 0, y = 1;
   return fib(x, y, n);
}

int main(){
   while (scanf("%lld %lld", &n, &MOD) != EOF){
       printf("%lld\n", fib(n));
   }
   return 0;
}
```

Fast Fibonacci LL.cpp

```
#include <bits/stdtr1c++.h>

#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/*** Fast Doubling Algorithm to calculate n'th fibonacci number ***/
/*** fib(0) = 0, fib(1) = 1, fib(n) = fib(n - 1) + fib(n - 2) ***/

/// Recursive version (Faster, at least in SPOJ)

int fib(int& x, int& y, long long n){
   if (!n) x = 0, y = 1;
   else{
       int c, d;
       fib(c, d, n >> 1);
       long long a = c, b = d, z = (d << 1) - c;
       if (z < 0) z += MOD;

       x = (a * z) % MOD;
       y = ((a * a) + (b * b)) % MOD;

       if (n & 1){
           x += y;
           if (x >= MOD) x -= MOD;
           swap(x, y);
       }
   }
   return x;
}

int fib(long long n){
   int x = 0, y = 1;
   return fib(x, y, n);
}

/// Iterative version

int fib(long long n){
   long long a = 0, b = 1, d;

   for (int i = 63 - __builtin_clzll(n); i >= 0; i--){
       d = (b << 1) - a;
       if (d < 0) d += MOD;

       d = (a * d) % MOD;
       b = ((a * a) + (b * b)) % MOD, a = d;
       if ((n >> i) & 1){
```

```
            d = a + b;
            if (d >= MOD) d -= MOD;
            a = b, b = d;
        }
    }

    return a;
}

int main(){
    int n;
    while (cin >> n) cout << fib(n) << endl;
}
```

Fast Fibonacci OP.cpp

```
#include <bits/stdtr1c++.h>

#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

namespace fib{ /// hash = 275701
    unsigned T[64][2][2], dp[8][256][2][2];

    inline unsigned mul(unsigned A[2][2], unsigned B[2][2]){
        unsigned C[2][2];
        C[0][0] = ((unsigned long long)A[0][0] * B[0][0] + (unsigned long
long)A[0][1] * B[1][0]) % MOD;
        C[0][1] = ((unsigned long long)A[0][0] * B[0][1] + (unsigned long
long)A[0][1] * B[1][1]) % MOD;
        C[1][0] = ((unsigned long long)A[1][0] * B[0][0] + (unsigned long
long)A[1][1] * B[1][0]) % MOD;
        C[1][1] = ((unsigned long long)A[1][0] * B[0][1] + (unsigned long
long)A[1][1] * B[1][1]) % MOD;
        A[0][0] = C[0][0], A[0][1] = C[0][1], A[1][0] = C[1][0], A[1][1] =
C[1][1];
    }

    /// n'th fibonacci number, 0, 1, 1, 2, 3, 5, 8, 13....
    inline unsigned fast_fibonacci(unsigned long long n){ /// 0.15 s for
10^6 random 64 bit numbers in Codeforces
        if (n <= 1) return n % MOD;

        n -= 2;
        unsigned i = 0, mask, ar[2][2] = {1, 0, 0, 1};
        while (n){
            mask = n & 255;
            if (mask) mul(ar, dp[i][mask]);
            i++, n >>= 8;
```

```
       }
       return (ar[0][0] + ar[0][1]) % MOD;
   }

   /// n and n + 1'th fibonacci number as pair, {0, 1}, {1, 1}, {1, 2},
{2, 3}....
   inline pair<int, int> fast_fibonacci2(unsigned long long n){
       if (n == 0) return make_pair(0, 1 % MOD);
       if (n == 1) return make_pair(1 % MOD, 1 % MOD);

       n--;
       unsigned i = 0, mask, ar[2][2] = {1, 0, 0, 1};
       while (n){
           mask = n & 255;
           if (mask) mul(ar, dp[i][mask]);
           i++, n >>= 8;
       }
       return make_pair((ar[1][0] + ar[1][1]) % MOD, (ar[0][0] +
ar[0][1]) % MOD);
   }

   inline void build(){ /// must call build() to initialize before
anything
       int i, j, k, l, mask;
       T[0][0][0] = 1, T[0][0][1] = 1, T[0][1][0] = 1, T[0][1][1] = 0;

       for (i = 1; i < 64; i++){
           T[i][0][0] = T[i - 1][0][0], T[i][0][1] = T[i - 1][0][1];
           T[i][1][0] = T[i - 1][1][0], T[i][1][1] = T[i - 1][1][1];
           mul(T[i], T[i]);
       }

       for (i = 0, j = 0; i < 64; j++, i += 8){
           dp[j][0][0][0] = 1, dp[j][0][0][1] = 0, dp[j][0][1][0] = 0,
dp[j][0][1][1] = 1;

           for (mask = 1; mask < 256; mask++){
               k = __builtin_ctz(mask), l = mask ^ (1 << k);
               dp[j][mask][0][0] = dp[j][l][0][0], dp[j][mask][0][1] =
dp[j][l][0][1];
               dp[j][mask][1][0] = dp[j][l][1][0], dp[j][mask][1][1] =
dp[j][l][1][1];
               mul(dp[j][mask], T[i + k]);
           }
       }
   }
}

int main(){
   int i, j, k, l, n, r;
   fib::build();

   for (i = 0; i <= 10; i++){
       pair <int, int> p = fib::fast_fibonacci2(i);
```

```
        printf("%d = %d %d\n", i, p.first, p.second);
    }

    n = 1000000;
    vector <long long> v;
    for (i = 0; i < n; i++){
        long long x = ran(1, 1000000000);
        long long y = ran(1, 1000000000);
        long long z = x << 31 ^ y;
        for (j = 0; j < 17; j++){
            k = ran(0, 62);
            z ^= (1LL << k);
        }

        v.push_back(z);
    }

    clock_t start = clock();
    unsigned long long h = 0;
    for (i = 0; i < v.size(); i++) h = (h * 1000000009) +
fib::fast_fibonacci(v[i]);

    dbg(h);
    fprintf(stderr, "%0.6f\n", (clock() - start) / (1.0 *
CLOCKS_PER_SEC));
    return 0;
}
```

Fast Fibonacci.cpp

```
#include <bits/stdtr1c++.h>

#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/*** Fast Doubling Algorithm to calculate n'th fibonacci number ***/
/*** fib(0) = 0, fib(1) = 1, fib(n) = fib(n - 1) + fib(n - 2) ***/

/// Recursive version (Faster, at least in SPOJ)

int fib(int& x, int& y, int n){
    if (!n) x = 0, y = 1;
    else{
        int c, d;
        fib(c, d, n >> 1);
        long long a = c, b = d, z = (d << 1) - c;
        if (z < 0) z += MOD;

        x = (a * z) % MOD;
        y = ((a * a) + (b * b)) % MOD;
```

```
        if (n & 1){
            x += y;
            if (x >= MOD) x -= MOD;
            swap(x, y);
        }
    }
    return x;
}

int fib(int n){
    int x = 0, y = 1;
    return fib(x, y, n);
}

/// Iterative version

int fib(int n){
    long long a = 0, b = 1, d;

    for (int i = 31 - __builtin_clz(n); i >= 0; i--){
        d = (b << 1) - a;
        if (d < 0) d += MOD;

        d = (a * d) % MOD;
        b = ((a * a) + (b * b)) % MOD, a = d;
        if ((n >> i) & 1){
            d = a + b;
            if (d >= MOD) d -= MOD;
            a = b, b = d;
        }
    }

    return a;
}

int main(){
    int n;
    while (cin >> n) cout << fib(n) << endl;
}
```

Fast Input Output.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <sys/mman.h>

#define clr(ar) (memset(ar, 0, sizeof(ar)))
#define read() freopen("lol.txt", "r", stdin)
#define parse(x) (x = (x * 10) + (str[j] - 48))
#define parse(j, x) (x = (x * 10) + (str[j] - 48))
#define parse(j, x) (x = (x << 1) + (x << 3) + (str[j] - 48))
```

```
int ye;
char temp[25], out[1000010];

void convert(int x){
   int i, r, d = 1;

   for (; ;){
       r = (x / 10);
       temp[d++] = (x - (r * 10)) + 48;
       x = r;
       if (!x) break;
   }

   for (i = d - 1; i; i--) out[ye++] = temp[i];
}

void convert(long long x){
   int i, d = 1;
   long long r;

   for (; ;){
       r = (x / 10);
       temp[d++] = (x - (r * 10)) + 48;
       x = r;
       if (!x) break;
   }

   for (i = d - 1; i; i--) out[ye++] = temp[i];
}



void convert(int x){
   int i, r, d = 1;

   for (; ;){
       r = (x * 0.1);
       temp[d++] = (x - (r * 10)) + 48;
       x = r;
       if (!x) break;
   }

   for (i = d - 1; i; i--) out[ye++] = temp[i];
}


char* mmap_in(){
   char* str = mmap(0, 6500000, PROT_READ, MAP_SHARED, fileno(stdin), 0);
   return str;
}

int main(){
   ye = 0;
   convert(123);
```

```
   fwrite(out, 1, ye, stdout);
}


#define BUFFER_SIZE 11

const char digits[] =
"000102030405060708091011121314151617181920212223242526272829303132333435
3637383940414243444546474849505152535455565758596061626364656667686970717
27374757677787980818283848586878889909192939495969798 99";

int ye = 0;
char buf[BUFFER_SIZE], out[100010];

void convert(int val){
   int c = 2, div = val / 100;
   char *it = (char*) &buf[BUFFER_SIZE - 2];

   while (div){
       memcpy(it, &digits[(val - (div * 100)) << 1], 2);
       it -= 2, c += 2;
       val = div;
       div = val / 100;
   }

   memcpy(it, &digits[val << 1], 2);
   if (val < 10) it++, c--;
   memcpy(&out[ye], it, c);
   ye += c;
}
```

Fast Input Output.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

namespace fastio{
   int ptr, ye;
   char temp[25], str[31333667], out[31333667];

   void init(){
       ptr = 0, ye = 0;
       fread(str, 1, 31333667, stdin);
   }

   inline long long number(){
       int i, j, neg = 0;
       long long val = 0;
```

```
        while (str[ptr] < 45 || str[ptr] > 57) ptr++;
        if (str[ptr] == 45) neg++, ptr++;
        while (str[ptr] > 47 && str[ptr] < 58) val = (val * 10) +
(str[ptr++] - 48);
        if (neg) val = -val;
        return val;
    }

    inline void convert(long long x){
        int i, d = 0;
        if (x < 0) x = -x, out[ye++] = 45;

        for (; ;){
            temp[++d] = (x % 10) + 48;
            x /= 10;
            if (!x) break;
        }

        for (i = d; i; i--) out[ye++] = temp[i];
    }

    inline void print(){
        fwrite(out, 1, ye, stdout);
    }

    inline void append(char ch){
        out[ye++] = ch;
    }
}

int main(){

}
```

Fast Integer Cube and Square Root.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

unsigned int fast_sqrt(unsigned int n){
    unsigned int c, g;

    c = g = 0x8000;
    for (; ;){
        if ((g * g) > n) g ^= c;
        c >>= 1;
        if (!c) return g;
        g |= c;
    }
}
```

```
int fast_cbrt(int n){
   int x, r = 30, res = 0;

   for (; r >= 0; r -= 3){
       res <<= 1;
       x = (3 * res * (res + 1)) + 1;
       if ((n >> r) >= x){
           res++;
           n -= (x << r);
       }
   }

   return res;
}

unsigned long long fast_sqrt(unsigned long long n){
   unsigned long long c, g;

   c = g = 0x80000000;
   for (; ;){
       if ((g * g) > n) g ^= c;
       c >>= 1;
       if (!c) return g;
       g |= c;
   }
}

unsigned long long fast_cbrt(unsigned long long n){
   int r = 63;
   unsigned long long x, res = 0;

   for (; r >= 0; r -= 3){
       res <<= 1;
       x = (res * (res + 1) * 3) + 1;
       if ((n >> r) >= x){
           res++;
           n -= (x << r);
       }
   }

   return res;
}

int main(){

}
```

Fast Sqrt.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
```

```
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

/* Machine Dependent + Not so fast when compared to math.h sqrt() */

float F(float y){
   long i;
   float x;
   const float threehalfs = 1.5F;

   x = y * 0.5F;
   i = *(long *) &y;
   i  = 0x5f3759df -(i >> 1);
   y  = *(float *) &i;

   y  = y * (threehalfs - (x * y * y )); /* Repeat this to increase
precision */
   y  = y * (threehalfs - (x * y * y ));
   y  = y * (threehalfs - (x * y * y ));
   y  = y * (threehalfs - (x * y * y ));
      return (1.0 / y);
}

double root(double y){
   double x;
   long long i;
   const double threehalfs = 1.5;

   x = y * 0.5;
   i = *(long long*) (&y);
   i = 0x5fe6ec85e7de30daLL - (i >> 1);
   y = *(double*) (&i);

   y = y * (1.5 - x * y * y); /* Repeat this to increase precision */
   y = y * (1.5 - x * y * y);
   y = y * (1.5 - x * y * y);
   y = y * (1.5 - x * y * y);
   return (1.0 / y);
}

int main(){
   printf("%0.9lf %0.9lf\n", F(25.0), root(25.0));
}
```

Faulhaber_s Formula (Custom Algorithm).cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 1010
#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
```

```
using namespace std;

namespace fool{
   #define MAXN 10000

   tr1::unordered_map <unsigned long long, int> mp;
   int inv, P[MAX], binomial[MAX][MAX], dp[MAXN][MAX];

   long long expo(long long x, long long n){
       x %= MOD;
       long long res = 1;

       while (n){
           if (n & 1) res = (res * x) % MOD;
           x = (x * x) % MOD;
           n >>= 1;
       }

       return (res % MOD);
   }

   void init(){
       int i, j;
       mp.clear();
       inv = expo(2, MOD - 2);

       P[0] = 1;
       for (i = 1; i < MAX; i++){
           P[i] = (P[i - 1] << 1);
           if (P[i] >= MOD) P[i] -= MOD;
       }

       for (i = 0; i < MAX; i++){
           for (j = 0; j <= i; j++){
               if (i == j || !j) binomial[i][j] = 1;
               else{
                   binomial[i][j] = (binomial[i - 1][j] + binomial[i -
1][j - 1]);
                   if (binomial[i][j] >= MOD) binomial[i][j] -= MOD;
               }
           }
       }

       for (i = 1; i < MAXN; i++){
           long long x = 1;
           for (j = 0; j < MAX; j++){
               dp[i][j] = dp[i - 1][j] + x;
               if (dp[i][j] >= MOD) dp[i][j] -= MOD;
               x = (x * i) % MOD;
           }
       }
   }

   /// Returns (1^k + 2^k + 3^k + .... n^k) % MOD
```

```
    long long F(unsigned long long n, int k){
        if (n < MAXN) return dp[n][k];

        if (n == 1) return 1;
        if (n == 2) return (P[k] + 1) % MOD;
        if (!k) return (n % MOD);
        if (k == 1){
            n %= MOD;
            return (((n * (n + 1)) % MOD) * inv) % MOD;
        }

        unsigned long long h = (n << 10LL) | k; /// Change hash function
according to limits of n and k
        long long res = mp[h];
        if (res) return res;

        if (n & 1) res = F(n - 1, k) + expo(n, k);
        else{
            long long m, z;
            m = n >> 1;
            res = (F(m, k) * P[k]) % MOD;

            m--, res++;
            for (int i = 0; i <= k; i++){
                z = (F(m, i) * binomial[k][i]) % MOD;
                z = (z * P[i]) % MOD;
                res += z;
            }
        }

        res %= MOD;
        return (mp[h] = res);
    }

    long long faulhaber(unsigned long long n, int k){
        ///fool::init();
        return F(n, k);
    }
}

int main(){
    fool::init();
    int t, i, j;
    long long n, k, res;

    cin >> t;
    while (t--){
        cin >> n >> k;
        res = fool::faulhaber(n, k);
        cout << res << endl;
    }
    return 0;
}
```

Faulhaber_s Formula 2.c

```
---------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 2510
#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int S[MAX][MAX], inv[MAX];

int expo(long long x, int n){
   x %= MOD;
   long long res = 1;

   while (n){
       if (n & 1) res = (res * x) % MOD;
       x = (x * x) % MOD;
       n >>= 1;
   }

   return (res % MOD);
}

void Generate(){
   int i, j;
   for (i = 0; i < MAX; i++) inv[i] = expo(i, MOD - 2);

   S[0][0] = 1;
   for (i = 1; i < MAX; i++){
       S[i][0] = 0;
       for (j = 1; j <= i; j++){
           S[i][j] = ( ((long long)S[i - 1][j] * j) + S[i - 1][j - 1]) %
MOD;
       }
   }
}

int faulhaber(long long n, int k){
   n %= MOD;
   if (!k) return n;

   int j;
   long long res = 0, p = 1;
   for (j = 0; j <= k; j++){
       p = (p * (n + 1 - j)) % MOD;
       res = (res + (((S[k][j] * p) % MOD) * inv[j + 1])) % MOD;
   }

   return (res % MOD);
}
```

```
int main(){
   Generate();
   printf("%d\n", faulhaber(1001212, 1000));
   return 0;
}
```

Faulhaber_s Formula 3.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 1010
#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int S[MAX][MAX];

void Generate(){ /// hash = 173404
   int i, j;

   S[0][0] = 1;
   for (i = 1; i < MAX; i++){
       S[i][0] = 0;
       for (j = 1; j <= i; j++){
           S[i][j] = ( ((long long)S[i - 1][j] * j) + S[i - 1][j - 1]) %
MOD;
       }
   }
}

long long faulhaber(long long n, int k){ /// hash = 766729
   if (!k) return (n % MOD);

   int i, j, l;
   long long z, res = 0;

   for (j = 0; j <= k; j++){
       z = 1;
       for (l = j + 1, i = 0; (i - 1) < j; i++){
           if ((l > 1) && !((n - i + 1) % l)){
               z = (z * (((n - i + 1) / l) % MOD) ) % MOD;
               l = 1;
           }
           else z = (z * ((n - i + 1) % MOD) ) % MOD;
       }
       res = (res + (z * S[k][j])) % MOD;
   }
   return res;
}

int main(){
   Generate();
```

```
    int t, k;
    long long n;

    scanf("%d", &t);
    while (t--){
        scanf("%lld %d", &n, &k);
        printf("%lld\n", faulhaber(n, k));
    }
    return 0;
}
```

Faulhaber_s Formula.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 2510
#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int S[MAX][MAX], inv[MAX], factorial[MAX], bernoulli[MAX];

int expo(long long x, int n){
    x %= MOD;
    long long res = 1;

    while (n){
        if (n & 1) res = (res * x) % MOD;
        x = (x * x) % MOD;
        n >>= 1;
    }

    return (res % MOD);
}

void Generate(){
    int i, j;
    long long x, y, z, lim = (long long)MOD * MOD;

    factorial[0] = 1;
    for (i = 1; i < MAX; i++){
        factorial[i] = ((long long) factorial[i - 1] * i) % MOD;
    }
    for (i = 0; i < MAX; i++) inv[i] = expo(i, MOD - 2);

    S[0][0] = 1;
    for (i = 1; i < MAX; i++){
        S[i][0] = 0;
        for (j = 1; j <= i; j++){
            S[i][j] = ( ((long long)S[i - 1][j] * j) + S[i - 1][j - 1]) %
MOD;
        }
```

```
    }

    bernoulli[0] = 1;
    for (i = 1; (i + 1) < MAX; i++){
        if ((i & 1) && i > 1) bernoulli[i] = 0;
        else{
            x = 0, y = 0;

            for (j = 0; j <= i; j++){
                z = ((long long) factorial[j] * inv[j + 1]) % MOD;
                z = (z * S[i][j]) % MOD;
                if (j & 1) y += z;
                else x += z;
            }
            bernoulli[i] = (lim + x - y) % MOD;

        }
    }
}

int faulhaber(long long n, int k){
    n %= MOD;
    if (!k) return n;

    n++, k++;
    int i, j;
    long long p = 1, res = n;

    for (i = 1; i < k; i++){
        p = (p * (k - i + 1)) % MOD;
        p = (p * inv[i]) % MOD;
        if (bernoulli[i]) res = (res + (p * bernoulli[i])) % MOD;
        res = (res * n) % MOD;
    }
    res = (res * inv[k]) % MOD;

    return res;
}

int main(){
    Generate();
    printf("%d\n", faulhaber(10, 4));
    return 0;
}
```

Fenwick Tree 2D + HashMap.cpp

```
--------------------------------------------------
/// 2D Fenwick Tree with HashMap
/// When N = 10^5, runs in 0.8 seconds in codeforces server for random
cases
/// When N = 2 * 10^5, runs in 2.00 seconds in codeforces server for
random cases

#include <bits/stdtr1c++.h>
```

```
#define MAX 200010
#define HMOD 16777215
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

int n, m, tree[HMOD + 71235];
long long hashmap[HMOD + 71235];

inline void add(int i, int j, int v){
   long long h = (((long long)i * 1000003) + j) + 917921;
   int k = h & HMOD;
   while (hashmap[k] && hashmap[k] != h) k++;
   hashmap[k] = h, tree[k] += v;
}

inline int find(int i, int j){
   long long h = (((long long)i * 1000003) + j) + 917921;
   int k = h & HMOD;
   while (hashmap[k] && hashmap[k] != h) k++;
   return (hashmap[k] ? tree[k] : 0);
}

/// Add v to ar[i][j]
inline void update(int i, int j, int v){
   while (i <= n){
       int k = j;
       while (k <= m){
           add(i, k, v);
           k += (k & (-k));
       }
       i += (i & (-i));
   }
}

/// Query for sum of the sub-rectangle from upper-left [i,j] to lower-
right [n,n]
inline int query(int i, int j){
   if ((i < 0) || (j < 0) || (i > n) || (j > m)) return 0;

   int res = 0;
   while (i){
       int k = j;
       while (k){
           res += find(i, k);
           k ^= (k & (-k));
       }
       i ^= (i & (-i));
   }
   return res;
```

```
}

/// Query for sum of the sub-rectangle from upper-left [i,j] to lower-
right [k,l]
inline int query(int i, int j, int k, int l){
   if (i > k || j > l) return 0;

   int res = query(k, l) - query(i - 1, l) + query(i - 1, j - 1) -
query(k, j - 1);
   return res;
}

int main(){
   clock_t start = clock();
   int q, i, j, k, l, x, y, z;

   n = 200000;
   m = n, q = n;

   long long res = 0;
   while (q--){
       int flag = ran(0, 1);

       if (flag == 0){
           i = ran(1, n), k = ran(i, n);
           j = ran(1, m), l = ran(j, m);
           res += query(i, j, k, l);
       }

       if (flag == 1){
           i = ran(1, n), j = ran(1, m), x = ran(1, 100);
           update(i, j, x);
       }
   }

   printf("%0.5f\n", (clock() - start) / (1.0 * CLOCKS_PER_SEC));
   return 0;
}
```

Fenwick Tree 2D.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 1010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/// 2D Fenwick tree, Point updates and range queries
class Fenwick2D{
   public:
   int n, m, tree[MAX][MAX];
```

```
   Fenwick2D(){
   }

   Fenwick2D(int a, int b){
       clr(tree);
       n = a, m = b;
   }

   /// Add v to ar[i][j]
   void update(int i, int j, int v){
       while (i <= n){
           int k = j;
           while (k <= m){
               tree[i][k] += v;
               k += (k & (-k));
           }
           i += (i & (-i));
       }
   }

   /// Query for sum of the sub-rectangle from upper-left [i,j] to lower-
right [n,n]
   int query(int i, int j){
       if ((i < 0) || (j < 0) || (i > n) || (j > m)) return 0;

       int res = 0;
       while (i){
           int k = j;
           while (k){
               res += tree[i][k];
               k ^= (k & (-k));
           }
           i ^= (i & (-i));
       }
       return res;
   }

   /// Query for sum of the sub-rectangle from upper-left [i,j] to lower-
right [k,l]
   int query(int i, int j, int k, int l){
       if (i > k || j > l) return 0;

       int res = query(k, l) - query(i - 1, l) + query(i - 1, j - 1) -
query(k, j - 1);
       return res;
   }
};

/// 2D Fenwick tree, Range updates and point queries
class Fenwick2D{
   public:
   int n, m, tree[MAX][MAX];
```

```
    Fenwick2D(){
    }

    Fenwick2D(int a, int b){
        clr(tree);
        n = a, m = b;
    }

    /// Add v to the sub-rectangle from upper-left [i,j] to lower-right
[n,n]
    void update(int i, int j, int v){
        if ((i < 0) || (j < 0) || (i > n) || (j > m)) return;

        while (i <= n){
            int k = j;
            while (k <= m){
                tree[i][k] += v;
                k += (k & (-k));
            }
            i += (i & (-i));
        }
    }

    /// Add v to the sub-rectangle from upper-left [i,j] to lower-right
[k,l]
    void update(int i, int j, int k, int l, int v){
        if (i > k || j > l) return;

        update(i, j, v);
        update(k + 1, j, -v);
        update(k + 1, l + 1, v);
        update(i, l + 1, -v);
    }

    /// Query for the value at index [i][j]
    int query(int i, int j){
        int res = 0;
        while (i){
            int k = j;
            while (k){
                res += tree[i][k];
                k ^= (k & (-k));
            }
            i ^= (i & (-i));
        }
        return res;
    }
};

/// 2D Fenwick tree, Range updates and range queries
class Fenwick2D{
    public:
    int n, m;
    long long tree[4][MAX][MAX];
```

```
    Fenwick2D(){
    }

    Fenwick2D(int a, int b){
        clr(tree);
        n = a, m = b;
    }

    /// Add v to the sub-rectangle from upper-left [p,q] to lower-right
[n,n]
    void update(int p, int q, long long v){
        if ((p < 0) || (q < 0) || (p > n) || (q > m)) return;

        int i = p, c = p - 1, d = q - 1;

        while (i <= n){
            int j = q;
            while (j <= m){
                tree[0][i][j] += v;
                tree[1][i][j] += (v * d);
                tree[2][i][j] += (v * c);
                tree[3][i][j] += (v * c * d);
                j += (j & (-j));
            }
            i += (i & (-i));
        }
    }

    /// Query for sum of the sub-rectangle from upper-left [p,q] to lower-
right [n,n]
    long long query(int p, int q){
        int i, j;
        long long x, y, z, c, d, res;

        i = p, x = y = z = 0;
        while (i){
            j = q, c = d = 0;
            while (j){
                c += tree[0][i][j];
                d += tree[1][i][j];
                y += tree[2][i][j];
                z += tree[3][i][j];
                j ^= (j & (-j));
            }
            i ^= (i & (-i));
            x += ((c * q) - d);
        }

        res = (x * p) - (y * q) + z;
        return res;
    }
```

```
   /// Add v to the sub-rectangle from upper-left [i,j] to lower-right
[k,l]
   void update(int i, int j, int k, int l, long long v){
       update(i, j, v);
       update(k + 1, j, -v);
       update(k + 1, l + 1, v);
       update(i, l + 1, -v);
   }

   /// Query for sum of the sub-rectangle from upper-left [i,j] to lower-
right [k,l]
   long long query(int i, int j, int k, int l){
       if (i > k || j > l) return 0;

       long long res = query(k, l) - query(i - 1, l) + query(i - 1, j -
1) - query(k, j - 1);
       return res;
   }
};
```

Fenwick Tree 3D.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 205
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/// 3D Fenwick tree, Range updates and point queries
struct Fenwick3D{
   int n, m, r, tree[MAX][MAX][MAX];

   Fenwick3D(){
   }

   Fenwick3D(int a, int b, int c){
       clr(tree);
       n = a, m = b, r = c;
   }

   /// Add v to the cube from lower-right [i,j,k] to upper-left [1,1,1]
   void update(int i, int j, int k, int v){
       if ((i < 0) || (j < 0) || (i > n) || (j > m) || (k < 0) || (k >
r)) return;

       while (i){
           int x = j;
           while (x){
               int y = k;
               while (y){
                   tree[i][x][y] += v;
```

```
                    y ^= (y & (-y));
                }
                x ^= (x & (-x));
            }
            i ^= (i & (-i));
        }
    }

    /// Add v to the cube from upper-left [x1,y1,z1] to lower-right
[x2,y2,z2]
    void update(int x1, int y1, int z1, int x2, int y2, int z2){
        update(x2, y2, z2, 1), update(x1 - 1, y1 - 1, z2, 1);
        update(x1 - 1, y2, z1 - 1, 1), update(x2, y1 - 1, z1 - 1, 1);
        update(x1 - 1, y2, z2, -1), update(x2, y1 - 1, z2, -1);
        update(x2, y2, z1 - 1, -1), update(x1 - 1, y1 - 1, z1 - 1, -1);
    }

    /// Query for the value at index [i][j][k]
    int query(int i, int j, int k){
        int res = 0;
        while (i <= n){
            int x = j;
            while (x <= m){
                int y = k;
                while (y <= r){
                    res += tree[i][x][y];
                    y += (y & (-y));
                }
                x += (x & (-x));
            }
            i += (i & (-i));
        }
        return res;
    }
};

int main(){
}
```

Fenwick Tree.cpp
```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/// Fenwick tree, Point updates and range queries
class Fenwick{
    public:
    int n, tree[MAX];
```

```
    Fenwick(){
    }

    Fenwick(int m){
        n = m;
        for (int i = 1; i <= n; i++) tree[i] = 0;
    }

    /// Add v to index p
    void update(int p, int v){
        while (p <= n){
            tree[p] += v;
            p += (p & (-p));
        }
    }

    /// Returns sum from index [1:p]
    int query(int p){
        int res = 0;
        while (p){
            res += tree[p];
            p ^= (p & (-p));
        }
        return res;
    }

    /// Returns sum from index [l:r]
    int query(int l, int r){
        if (l > r) return 0;
        return (query(r) - query(--l));
    }
};

/// Fenwick tree, Range updates and point queries
class Fenwick{
    public:
    int n, tree[MAX];

    Fenwick(){
    }

    Fenwick(int m){
        n = m;
        for (int i = 1; i <= n; i++) tree[i] = 0;
    }

    /// Add v to index [p:n]
    void update(int p, int v){
        while (p <= n){
            tree[p] += v;
            p += (p & (-p));
        }
    }
```

```
    /// Add v to index [l:r]
    void update(int l, int r, int v){
        if (l > r) return;
        update(l, v);
        update(r + 1, -v);
    }

    /// Returns value of index p
    int query(int p){
        int res = 0;
        while (p){
            res += tree[p];
            p ^= (p & (-p));
        }
        return res;
    }
};

/// Fenwick tree, Range updates and range queries
class Fenwick{
    public:
    int n;
    long long tree[MAX], temp[MAX];

    Fenwick(){
    }

    Fenwick(int m){
        n = m;
        for (int i = 1; i <= n; i++) tree[i] = temp[i] = 0;
    }

    void update(long long* tree, int p, long long v){
        while (p <= n){
            tree[p] += v;
            p += (p & (-p));
        }
    }

    /// Add v to index [l:r]
    void update(int l, int r, long long v){
        if (l > r) return;
        update(tree, l, v);
        update(tree, r + 1, -v);
        update(temp, l, v * (l - 1));
        update(temp, r + 1, (-v) * r);
    }

    long long query(long long* tree, int p){
        long long res = 0;
        while (p){
            res += tree[p];
            p ^= (p & (-p));
```

```
        }
        return res;
    }

    /// Returns sum from index [1:p]
    long long query(int p){
        return ((query(tree, p) * p) - query(temp, p));
    }

    /// Returns sum from index [l:r]
    long long query(int l, int r){
        if (l > r) return 0;
        return (query(r) - query(--l));
    }
};

int main(){

}
```

Fractions Class.cpp

```
------------------------------------------------
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

struct Fraction{
    long long numerator, denominator;

    long long gcd(long long a, long long b){
        while (b) b ^= a ^= b ^= a %= b;
        return a;
    }

    Fraction(){}

    Fraction(long long x, long long y){
        long long g = gcd(abs(x), abs(y));
        numerator = x / g;
        denominator = y / g;
    }

    Fraction operator+ (Fraction);
    Fraction operator- (Fraction);
    Fraction operator* (Fraction);
    Fraction operator/ (Fraction);
};

Fraction Fraction::operator+ (Fraction param){
```

```
   long long x = (numerator * param.denominator) + (denominator *
param.numerator);
   long long y = (denominator * param.denominator);
   return Fraction(x, y);
}

Fraction Fraction::operator- (Fraction param){
   long long x = (numerator * param.denominator) - (denominator *
param.numerator);
   long long y = (denominator * param.denominator);
   return Fraction(x, y);
}

Fraction Fraction::operator* (Fraction param){
   long long x = (numerator * param.numerator);
   long long y = (denominator * param.denominator);
   return Fraction(x, y);
}

Fraction Fraction::operator/ (Fraction param){
   long long x = (numerator * param.denominator);
   long long y = (denominator * param.numerator);
   return Fraction(x, y);
}

int main(){

}
```

GCD.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int gcd(int u, int v){
   int t;

   while (v){
       t = u;
       u = v;
       v = t % v;
   }

   if (u < 0) u = 0 - u;
   return u;
}

long long gcd(long long u, long long v){
   long long t;
```

```
    while (v){
        t = u;
        u = v;
        v = t % v;
    }

    if (u < 0) u = 0 - u;
    return u;
}

int gcd(int a, int b){
    while (b) b ^= a ^= b ^= a %= b;
    return a;
}

long long gcd(long long a, long long b){
    while (b) b ^= a ^= b ^= a %= b;
    return a;
}

unsigned long long gcd(unsigned long long u, unsigned long long v){
    if (!u) return v;
    if (!v) return u;
    if (u == 1 || v == 1) return 1;

    int shift = __builtin_ctzll(u | v);
    u >>= __builtin_ctzll(u);
    do{
        v >>= __builtin_ctzll(v);
        if (u > v)
            v ^= u ^= v ^= u;
        v = v - u;
    } while (v);

    return u << shift;
}

unsigned int gcd(unsigned int u, unsigned int v){
    if (!u) return v;
    if (!v) return u;
    if (u == 1 || v == 1) return 1;

    int shift = __builtin_ctz(u | v);
    u >>= __builtin_ctz(u);
    do{
        v >>= __builtin_ctz(v);
        if (u > v)
            v ^= u ^= v ^= u;
        v = v - u;
    } while (v);

    return u << shift;
}
```

```
int gcd(int u, int v){
   if (!u || !v) return (u | v);

   int d, shift;
   for (shift = 0; !((u | v) & 1); shift++){
       u >>= 1;
       v >>= 1;
   }

   while (!(u & 1)) u >>= 1;
   do{
       while (!(v & 1)) v >>= 1;
       if (u < v) v -= u;
       else{
           d = u - v;
           u = v;
           v = d;
       }
       v >>= 1;
   }
   while (v);

   return (u << shift);
}

int gcd(int a, int b){
   if (!a) return b;
   else return gcd(b, a % b);
}

long long gcd(long long a, long long b){
   if (!a) return b;
   else return gcd(b, a % b);
}

int main(){
}
```

Gambler_s Ruin.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

long double expo(long double x, int n){
   if (n == 0) return 1;
   if (n & 1) return expo(x, n - 1) * x;
   else{
       long double res = expo(x, n >> 1);
       return res * res;
   }
```

```
}

/// Gambler's ruin problem
/// First player has n1 coins, second player has n2 coins
/// After each move, first player wins with probability p and second
player wins with probability q (p + q == 1)
/// The loser gives 1 coin to the winner
/// When number of coins reaches 0, a player loses
/// Returns the probability of first player winning

long double gamblers_ruin(int n1, int n2, long double p, long double q){
   if (fabs(p - q) <= 1e-9){
       return (long double)n2 / (n1 + n2);
   }

   long double x = 1.0 - expo(q / p, n2);
   long double y = 1.0 - expo(q / p, n1 + n2);
   return (x / y);
}

int main(){

}
```

Gaussian Elimination Extended.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define EPS 1e-9
#define MAXROW 512
#define MAXCOL 512
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/***

Gauss-Jordan Elimination

n = number of linear equations
m = number of variables
ar[i][m] = right-hand side value of constants

For instance, the system of linear equations becomes:

2x + y -z = 8       ----->  (i)
-3x -y + 2z = -11   ----->  (ii)
-2x + y + 2z = -3   ----->  (iii)

n = 3 (x, y, z), m = 3 (i, ii, iii)
ar[0] = {2, 1, -1, 8}     ----->  (i)
```

```
ar[1] = {-3, -1, 2, -11} ----->  (ii)
ar[2] = {-2, 1, 2, -3}   ----->  (iii)


Returns -1 when there is no solution
Otherwise returns the number of independent variables (0 for an unique
solution)
Contains a solution in the vector res on successful completion
Note that the array is modified in the process

Notes:
For solving problems on graphs with probability/expectation, make sure
the graph
is connected and a single component. If not, then re-number the vertex
and solve
for each connected component separately.

***/

int gauss(int n, int m, double ar[MAXROW][MAXCOL], vector<double>& res){
/// hash = 835176
    res.assign(m, 0);
    vector <int> pos(m, -1);
    int i, j, k, l, p, free_var = 0;

    for (j = 0, i = 0; j < m && i < n; j++){
        for (k = i, p = i; k < n; k++){
            if (abs(ar[k][j]) > abs(ar[p][j])) p = k;
        }

        if (abs(ar[p][j]) > EPS){
            pos[j] = i;
            for (l = j; l <= m; l++) swap(ar[p][l], ar[i][l]);

            for (k = 0; k < n; k++){
                if (k != i){
                    double x = ar[k][j] / ar[i][j];
                    for (l = j; l <= m; l++) ar[k][l] -= (ar[i][l] * x);
                }
            }
            i++;
        }
    }

    for (i = 0; i < m; i++){
        if (pos[i] == -1) free_var++;
        else res[i] = ar[pos[i]][m] / ar[pos[i]][i];
    }

    for (i = 0; i < n; i++) {
        double val = 0.0;
        for (j = 0; j < m; j++) val += (res[j] * ar[i][j]);
        if (abs(val - ar[i][m]) > EPS) return -1;
    }
```

```
    return free_var;
}

int main(){

}
```

Gaussian Elimination On Band Matrix.c

```
/// Gaussian Elimination on Sparse Band Matrix (Non-zero values confined
to diagonal bands)
/// Complexity: O(n * m^3), n = number of rows, m = number of columns

#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAXM 22
#define MAXN 10002
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define cell_num(i, j) (m * (n - (i)) - (j) - 1)
#define valid(i, j) ((i) >= 0 && (i) < n && (j) >= 0 && (j) < m)

const int dx[] = {1, 0, -1, 0};
const int dy[] = {0, 1, 0, -1};

int n, m;
double band[MAXN * MAXM][2 * MAXM + 1]; /// sparse band matrix, used to
compactly represent non-zero entries of the gauss matrix
double rhs[MAXN * MAXM], aux[MAXN * MAXM]; /// rhs[] contains right-hand
side constants before gauss, and solutions for the system after gauss

void init(int n, int m){
    int i, j, q = n * m, d = 2 * m + 1;

    clr(rhs);
    for (i = 0; i < q; i++){
        for (j = 0; j < d; j++){
            band[i][j] = 0.0;
        }
        band[i][m] = 1.0;
    }
}

/// adding p to gauss_matrix[r][c] (implicit) where r = cell_num[i][j]
and c = cell_num[k][l]
/// gauss_matrix[][] does not exist, the co-ordinates are converted to
band matrix representation
void add(int i, int j, int k, int l, double p){
    int u = cell_num(i, j), v = cell_num(k, l) - u + m;
    band[u][v] += p;
}
```

```
void gauss(int n, int m){
   double x, y;
   int i, j, k, l, d, u, v, q = n * m, r = 2 * m + 1;

   /// Forward Elimination
   for (i = 0; i < n; i++){
       for (j = 0; j < m; j++){
           d = i * m + j;
           x = band[d][m], rhs[d] /= x;
           for (k = 0; k <= m && (d + k) < q; k++) band[d][m + k] /= x;

           for (l = 1; l <= m && (d + l) < q; l++){
               x = band[d + l][m - l], rhs[d + l] -= (x * rhs[d]);
               for (k = 0; k <= m && (d + k) < q; k++){
                   band[d + l][m - l + k] -= (x * band[d][m + k]);
               }
           }
       }
   }

   /// Backward substitution
   for (i = 0; i < q; i++) aux[i] = rhs[i], rhs[i] = 0.0;
   for (i = 0; i < n; i++){
       for (j = 0; j < m; j++){
           x = 0.0, u = cell_num(i, j);
           for (v = 0; v < r; v++) x += (band[u][v] * rhs[v + u - m]);
           rhs[u] = aux[u] - x;
       }
   }
}

int main(){
   scanf("%d %d", &n, &m);
   init(n, m);
   return 0;
}
```

Gaussian Elimination in GF(2).cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAXROW 111
#define MAXCOL 111
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/***

Gauss-Jordan Elimination in Galois Field, GF(2)
```

```
n = number of linear equations
m = number of variables
ar[i][m] = right-hand side value of constants

For instance, the system of linear equations (note not in GF(2)) becomes:

2x + y -z = 8      ----->  (i)
-3x -y + 2z = -11  ----->  (ii)
-2x + y + 2z = -3  ----->  (iii)

n = 3 (x, y, z), m = 3 (i, ii, iii)
ar[0] = {2, 1, -1, 8}    ----->  (i)
ar[1] = {-3, -1, 2, -11} ----->  (ii)
ar[2] = {-2, 1, 2, -3}   ----->  (iii)


Returns -1 when there is no solution
Otherwise returns the number of independent variables (0 for an unique
solution)
Contains a solution in the bit vector res on successful completion
Note that the array is modified in the process

***/

int gauss(int n, int m, bitset <MAXCOL> ar[MAXROW], bitset <MAXCOL>&
res){ /// hash = 169721
   res.reset();
   vector <int> pos(m, -1);
   int i, j, k, l, v, p, free_var = 0;

   for (j = 0, i = 0; j < m && i < n; j++){
       for (k = i, p = i; k < n; k++){
           if (ar[k][j]){
               p = k;
               break;
           }
       }

       if (ar[p][j]){
           pos[j] = i;
           swap(ar[p], ar[i]);

           for (k = 0; k < n; k++){
               if (k != i && ar[k][j]) ar[k] ^= ar[i];
           }
           i++;
       }
   }

   for (i = 0; i < m; i++){
       if (pos[i] == -1) free_var++;
       else res[i] = ar[pos[i]][m];
   }
```

```
    for (i = 0; i < n; i++) {
        for (j = 0, v = 0; j < m; j++) v ^= (res[j] & ar[i][j]);
        if (v != ar[i][m]) return -1;
    }

    return free_var;
}

int main(){

}
```

Gaussian Elimination in Modular Field.cpp

```
#include <bits/stdtr1c++.h>

#define MAXROW 1010
#define MAXCOL 1010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

int expo(int a, int b, int MOD){
    int res = 1;

    while (b){
        if (b & 1) res = (long long)res * a % MOD;
        a = (long long)a * a % MOD;
        b >>= 1;
    }
    return res;
}

/// Gaussian elimination in field MOD (MOD should be a prime)
int gauss(int n, int m, int MOD, int ar[MAXROW][MAXCOL], vector<int>&
res){
    res.assign(m, 0);
    vector <int> pos(m, -1);
    int i, j, k, l, p, d, free_var = 0;
    const long long MODSQ = (long long)MOD * MOD;

    for (j = 0, i = 0; j < m && i < n; j++){
        for (k = i, p = i; k < n; k++){
            if (ar[k][j] > ar[p][j]) p = k;
        }

        if (ar[p][j]){
            pos[j] = i;
            for (l = j; l <= m; l++) swap(ar[p][l], ar[i][l]);
```

```
            d = expo(ar[i][j], MOD - 2, MOD);
            for (k = 0; k < n && d; k++){
                if (k != i && ar[k][j]){
                    int x = ((long long)ar[k][j] * d) % MOD;
                    for (l = j; l <= m && x; l++){
                        if (ar[i][l]) ar[k][l] = (MODSQ + ar[k][l] -
((long long)ar[i][l] * x)) % MOD;
                    }
                }
            }
            i++;
        }
    }

    for (i = 0; i < m; i++){
        if (pos[i] == -1) free_var++;
        else res[i] = ((long long)ar[pos[i]][m] * expo(ar[pos[i]][i], MOD
- 2, MOD)) % MOD;
    }

    for (i = 0; i < n; i++) {
        long long val = 0;
        for (j = 0; j < m; j++) val = (val + ((long long)res[j] *
ar[i][j])) % MOD;
        if (val != ar[i][m]) return -1;
    }
    return free_var;
}

int main(){

}
```

Gaussian Elimination.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 105
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/***

n = number of unknown variables
ar[i][n] = right-hand side values of constants

ar[0][0] * x1 + ar[0][1] * x2 + ... ar[0][n - 1] * xn = ar[0][n]
ar[1][0] * x1 + ar[1][1] * x2 + ... ar[1][n - 1] * xn = ar[1][n]
....
ar[n - 1][0] * x1 + ar[n - 1][1] * x2 + ... ar[n - 1][n - 1] * xn = ar[n
- 1][n]
```

```
After running gauss,
x1 = ar[0][n], x2 = ar[1][n].....xn = ar[n - 1][n]
Also, R[0] = x1, R[1] = x2 ..... R[n - 1] = xn (R is used just for
convenience)


Notes:
For solving problems on graphs with probability/expectation, make sure
the graph
is connected and a single component. If not, then re-number the vertex
and solve
for each connected component separately.

***/

bool gauss(int n, double ar[MAX][MAX], double* R){
   int i, j, k, r;

   for (i = 0; i < n; i++){
       r = i;
       for (j = i + 1; j < n; j++){
           if (abs(ar[j][i]) > abs(ar[r][i])) r = j;
       }
       if (abs(ar[r][i]) < 1e-9) return false; /// No unique solution
exists, set EPS carefully

       for (j = 0; j <= n && r != i; j++) swap(ar[i][j], ar[r][j]);
       for (k = i + 1; k < n; k++){
           double x = ar[k][i] / ar[i][i];
           for (j = i; j <= n; j++){
               ar[k][j] -= (x * ar[i][j]);
           }
       }
   }

   for (i = n - 1; i >= 0; i--){
       for (j = i + 1; j < n; j++){
           ar[i][n] -= (ar[j][n] * ar[i][j]);
       }
       ar[i][n] /= ar[i][i];
       R[i] = ar[i][n];
   }

   return true;
}

int main(){
}

Geometry.cpp
--------------------------------------------------
#include <bits/stdtr1c++.h>
```

```
#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

struct Point{
   long long x, y;

   Point(){
   }

   Point(long long xi, long long yi){
       x = xi, y = yi;
   }
};

struct Segment{
   struct Point P1, P2;

   Segment(){
   }

   Segment(struct Point P1i, struct Point P2i){
       P1 = P1i, P2 = P2i;
   }
};

/// Returns 0 if ABC is collinear, positive if ABC is a left turn,
negative if ABC is a right turn
long long ccw(struct Point A, struct Point B, struct Point C){
   return ((B.x - A.x) * (C.y - A.y)) - ((C.x - A.x) * (B.y - A.y));
}

/// Returns the shortest distance from Segment S to Point P
double dis(struct Segment S, struct Point P){
   double p, xx, yy;
   long long x = P.x, y = P.y, x1 = S.P1.x, y1 = S.P1.y, x2 = S.P2.x, y2
= S.P2.y;
   long long a = x - x1, b = y - y1, c = x2 - x1, d = y2 - y1, dot = (a *
c) + (b * d), len = (c * c) + (d * d);

   if ((dot < 0) || (x1 == x2 && y1 == y2)) xx = x1, yy = y1;
   else if (dot > len) xx = x2, yy = y2;
   else p = (1.0 * dot) / len, xx = x1 + (p * c), yy = y1 + (p * d);
   xx = -xx + x, yy = -yy + y;
   return sqrt((xx * xx) + (yy * yy));
}

/// Returns true if Point P lies on the Segment (both end-points
inclusive)
```

```
bool PointOnSeg(struct Segment S, struct Point P){
   long long x = P.x, y = P.y, x1 = S.P1.x, y1 = S.P1.y, x2 = S.P2.x, y2
= S.P2.y;
   long long a = x - x1, b = y - y1, c = x2 - x1, d = y2 - y1, dot = (a *
c) + (b * d), len = (c * c) + (d * d);

   if (x1 == x2 && y1 == y2) return (x1 == x && y1 == y);
   if (dot < 0 || dot > len) return false;
   return ((((x1 * len) + (dot * c)) == (x * len)) && (((y1 * len) + (dot
* d)) == (y * len)));
}

struct Polygon{
   #define CLOCKWISE 11230926
   #define ANTICLOCK 37281927

   int n; /// n should be greater than 1
   struct Point ar[MAX]; /// Points in polygon in clockwise order

   Polygon(){
   }

   /// Points in T are either in anti-clockwise or clockwise order
   Polygon(int ni, struct Point* T, int flag){
       n = ni;
       for (int i = 0; i < n; i++){
           if (flag == CLOCKWISE) ar[i] = T[i];
           else ar[i] = T[n - i - 1];
       }
   }

   /// strictly should be true if P needs to be strictly contained in the
polygon
   bool contains(struct Point P, bool strictly){
       int mid, low = 1, high = n - 1, idx = 1;

       while (low <= high){
           mid = (low + high) >> 1;
           if (ccw(ar[0], P, ar[mid]) > 0) low = mid + 1;
           else idx = mid, high = mid - 1;
       }

       if (!strictly && PointOnSeg(Segment(ar[0], ar[n - 1]), P)) return
true;
       if (!strictly && PointOnSeg(Segment(ar[idx], ar[idx - 1]), P))
return true;
       if (idx == 1 || ccw(ar[0], P, ar[n - 1]) == 0) return false;
       return (ccw(ar[idx], P, ar[idx - 1]) < 0);
   }

   double area(){
       if (n < 3) return 0.0;

       double res = 0.0;
```

```
        for (int i = 0, j = n - 1; i < n; j = i++) res += (ar[j].x *
ar[i].y) - (ar[j].y * ar[i].x);
        return fabs(res) / 2.0;
    }
};

int main(){
    return 0;
}
```

Gradient.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

typedef pair<int, int> Pair;

Pair filter(Pair p){ /// Normalize the gradient in P, gradient = (y / x)
= (P.second - P.first)
    if (p.first < 0){
        p.first *= -1;
        p.second *= -1;
    }
    else if (!p.first && p.second < 0) p.second *= -1;

    int g = __gcd(abs(p.first), p.second);
    return Pair(p.first / g, p.second / g);
}

/// Segments A-B and C-D are collinear if and only if the values returned
by this function
/// for both segments are equivalent

Pair G(Pair X, Pair Y){ /// Returns the gradient of the segment X-Y
    int y = Y.second - X.second;
    int x = Y.first - X.first;
    return filter(Pair(x, y));
}

int main(){
}
```

Gray Codes.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <assert.h>
```

```
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

long long gray_code(long long x){
   return x ^ (x >> 1);
}

long long inverse_gray_code(long long x){
   long long h = 1, res = 0;
   do{
       if (x & 1) res ^= h;
       x >>= 1, h = (h << 1) + 1;
   } while (x);
   return res;
}

int main(){

}
```

Great Circle Distance.c

```
--------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>
#include <assert.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

const long double PI = 2.0 * acos(0.0);

/// latitude and longitude of 2 points in degrees and radius of earth
long double haversine(long double lat1, long double lon1, long double
lat2, long double lon2, long double radius){
   lat1 = (lat1 * PI) / 180.0;
   lat2 = (lat2 * PI) / 180.0;
   lon1 = (lon1 * PI) / 180.0;
   lon2 = (lon2 * PI) / 180.0;

   return radius * acos(sin(lat1) * sin(lat2) + cos(lon2 - lon1) *
cos(lat1) * cos(lat2));
}

int main(){

}
```

Greedy Coin Change.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
```

```
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

const int n = 10;
const long long INF = 0x7FFFFFFFFFFFFFFF;
const int coins[] = {0, 1, 5, 10, 20, 50, 100, 200, 500, 1000, 2000}; ///
1 based indexing for coins

long long res, counter[44], sum[44]; /// counter[i] = number of coins of
type i

void backtrack(int i, long long p, long long c){ /// Complexity 2^n
   if (p == 0 && c < res) res = c; /// Change min to max here
   if (i == 0 || p <= 0) return;

   long long k, x = 0;
   if ((p - sum[i - 1]) > x) x = p - sum[i - 1];
   k = (x / coins[i]) + (x % coins[i] != 0);
   if (k <= counter[i]) backtrack(i - 1, p - k * coins[i], c + k);
   if (++k <= counter[i]) backtrack(i - 1, p - k * coins[i], c + k); ///
Do this multiple times if WA (around 5 or 10 should do)
}

/// Minimum number of coins required to make s from coin values and count
of coin values
/// -1 if no solution

long long solve(long long s){
   int i, j;
   for (i = 1; i <= n; i++) sum[i] = sum[i - 1] + coins[i] * counter[i];

   res = INF;
   backtrack(n, s, 0);
   if (res == INF) res = -1;
   return res;
}

int main(){
   return 0;
}
```

HLD + RMQ On Edges.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

#define NIL 0
#define MAX 50010
#define OPT(a, b) ((a)+(b))
#define jump(x) ((num[x] == 0) ? -1 : down[up[x]])
```

```
using namespace std;

/// Heavy Light Decomposition on Trees, 0 based indices
/// With RMQ support for edges
/// Define the operation, default is +
/// x * NIL = x, NIL = 0 for addition/subtraction, 1 for multiplication,
INF/-INF for min/max, etc
/// RMQ to add values on edges, if required to set/replace values modify
appropriately

namespace hld{ /// hash = 163953
   int r, n, id;
   vector <int> adj[MAX], weight[MAX];
   int nodeval[MAX], lazy[4 * MAX], tree[4 * MAX]; /// RMQ
   int parent[MAX], children[MAX], height[MAX], num[MAX], up[MAX],
down[MAX]; /// HLD

   /// num[i] = 0 if the edge from i to parent[i] is not heavy, otherwise
num[i] = unique id of the heavy edge
   /// down[i] = -1 if there is no heavy edge from i to it's children,
otherwise down[i] = node number of the heavy child of i
   /// up[i] = i, if i is root, otherwise up[i] = node number of parent
of i following only heavy up edges and one last light edge

   void dfs(int i, int p){
       parent[i] = p, children[i] = 1;
       if (i != p) height[i] = height[p] + 1;

       int j, x, len = adj[i].size();
       for (j = 0; j < len; j++){
           x = adj[i][j];
           if (x != p){
               dfs(x, i);
               nodeval[x] = weight[i][j];
               children[i] += children[x];
           }
       }
   }

   /// build heavy light decomposition
   void build(int i, int p){ /// hash = 989687
       up[i] = i;
       if (num[i]) up[i] = up[p];

       int j, x, h = -1, l = adj[i].size();
       for (j = 0; j < l; j++){
           x = adj[i][j];
           if ((x != p) && ((children[x] << 1) >= children[i])) h = x;
       }

       if (h != -1){
           num[h] = ++id;
           build(h, i);
```

```
        }
        for (j = 0, down[i] = h; j < l; j++){
            x = adj[i][j];
            if ((x != p) && (x != h)) build(x, i);
        }
    }

    void update_rmq(int idx, int a, int b, int l, int r, int x); /// RMQ
update defined for build
    void build(int root){
        r = root, id = 0;
        height[r] = 0, nodeval[r] = NIL;

        dfs(r, r);
        build(r, r);
        for (int i = 0; i < n; i++){
            if (up[i] == i) up[i] = parent[i];
        }

        /// Builds RMQ
        clr(lazy);
        for (int i = 0; i < (MAX << 2); i++) tree[i] = NIL;
        for (int i = 0; i < n; i++){
            if (num[i]) update_rmq(1, 1, id, num[i], num[i], nodeval[i]);
        }
    }
    void build(){
        build(0); /// Root set to 0 by default!
    }

    int lca(int a, int b){
        while (up[a] != up[b]){
            if (height[up[a]] > height[up[b]]) a = up[a];
            else b = up[b];
        }

        if (a == b) return a;
        if (num[a] && num[b]){
            if (height[a] < height[b]) return a;
            else return b;
        }
        return up[a];
    }

    void add_edge(int a, int b, int w){
        adj[a].push_back(b), weight[a].push_back(w);
        adj[b].push_back(a), weight[b].push_back(w);
    }

    void init(int nodes){
        clr(num), n = nodes;
        for (int i = 0; i < MAX; i++) adj[i].clear(), weight[i].clear();
    }
```

```
/************** RMQ functions **************/


/// Change lazy propagation accordingly
/// Note lazy and updates set for adding values in node, update if
set/replace operation
inline void push(int idx, int a, int b){
    int c = (a + b) >> 1, d = c + 1, p = idx << 1, q = p | 1;
    if (lazy[idx]){
        tree[idx] += (lazy[idx] * (b - a + 1)); /// Change lazy
according to operation
        if (a != b) lazy[p] += lazy[idx], lazy[q] += lazy[idx]; ///
Change lazy according to operation
        lazy[idx] = 0;
    }
}

/// Change query accordingly
int query_rmq(int idx, int a, int b, int l, int r){
    int c = (a + b) >> 1, d = c + 1, p = idx << 1, q = p | 1;

    push(idx, a, b);
    if (a == l && b == r) return tree[idx];
    else if (r <= c) return query_rmq(p, a, c, l, r);
    else if (l >= d) return query_rmq(q, d, b, l, r);
    else return OPT(query_rmq(p, a, c, l, c), query_rmq(q, d, b, d,
r));
}

/// Change update accordingly
void update_rmq(int idx, int a, int b, int l, int r, int x){ /// hash
= 487503
    int p = (idx << 1), q = p + 1, c = (a + b) >> 1, d = c + 1;

    if (a == l && b == r) lazy[idx] += x; /// Change lazy according to
operation, change here if set
    push(idx, a, b);
    if (a == l && b == r) return;

    if (r <= c){
        push(q, d, b);
        update_rmq(p, a, c, l, r, x);
    }
    else if (l >= d){
        push(p, a, c);
        update_rmq(q, d, b, l, r, x);
    }
    else{
        update_rmq(p, a, c, l, c, x);
        update_rmq(q, d, b, d, r, x);
    }

    tree[idx] = OPT(tree[p], tree[q]);
}
```

```
    /************** HLD + RMQ **************/

    /// Sum of all edges in the path from u to l, l must be an ancestor of
u
    int query_tree(int u, int l){ /// hash = 486879
        int res = NIL;
        while (height[u] > height[l]){
            if (num[u]){
                int v = jump(u);
                if (height[v] <= height[l]) v = down[l];
                res = OPT(res, query_rmq(1, 1, id, num[v], num[u]));
                u = parent[v];
            }
            else res = OPT(nodeval[u], res), u = parent[u];
        }
        return res;
    }

    /// Sum of all edges in the path from u to v
    int query(int u, int v){
        int l = lca(u, v), res = NIL;
        res = OPT(res, query_tree(u, l));
        res = OPT(res, query_tree(v, l));
        return res;
    }

    /// Add w to all edges in the path from u to l, l must be an ancestor
of u
    void update_tree(int u, int l, int w){
        while (height[u] > height[l]){
            if (num[u]){
                int v = jump(u);
                if (height[v] <= height[l]) v = down[l];
                update_rmq(1, 1, id, num[v], num[u], w);
                u = parent[v];
            }
            else nodeval[u] = OPT(nodeval[u], w), u = parent[u]; ///
Change here if set instead of add
        }
    }

    /// Add w to all edges in the path from u to v
    void update(int u, int v, int w){
        int l = lca(u, v);
        update_tree(u, l, w);
        update_tree(v, l, w);
    }
}

int main(){

}
```

HLD + RMQ On Nodes.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

#define NIL 0
#define MAX 50010
#define OPT(a, b) ((a)+(b))
#define jump(x) ((num[x] == 0) ? -1 : down[up[x]])

using namespace std;

/// Heavy Light Decomposition on Trees, 0 based indices
/// With RMQ support for nodes
/// Define the operation, default is +
/// x * NIL = x, NIL = 0 for addition/subtraction, 1 for multiplication,
INF/-INF for min/max, etc
/// RMQ to add values on nodes, if required to set/replace values modify
appropriately

namespace hld{
   int r, n, id;
   vector <int> adj[MAX];
   int nodeval[MAX], lazy[4 * MAX], tree[4 * MAX]; /// RMQ
   int parent[MAX], children[MAX], height[MAX], num[MAX], up[MAX],
down[MAX]; /// HLD

   /// num[i] = 0 if the edge from i to parent[i] is not heavy, otherwise
num[i] = unique id of the heavy edge
   /// down[i] = -1 if there is no heavy edge from i to it's children,
otherwise down[i] = node number of the heavy child of i
   /// up[i] = i, if i is root, otherwise up[i] = node number of parent
of i following only heavy up edges and one last light edge

   void dfs(int i, int p){
       parent[i] = p, children[i] = 1;
       if (i != p) height[i] = height[p] + 1;

       int j, x, len = adj[i].size();
       for (j = 0; j < len; j++){
           x = adj[i][j];
           if (x != p){
               dfs(x, i);
               children[i] += children[x];
           }
       }
   }

   /// build heavy light decomposition
   void build(int i, int p){ /// hash = 989687
       up[i] = i;
```

```
        if (num[i]) up[i] = up[p];

        int j, x, h = -1, l = adj[i].size();
        for (j = 0; j < l; j++){
            x = adj[i][j];
            if ((x != p) && ((children[x] << 1) >= children[i])) h = x;
        }

        if (h != -1){
            num[h] = ++id;
            build(h, i);
        }
        for (j = 0, down[i] = h; j < l; j++){
            x = adj[i][j];
            if ((x != p) && (x != h)) build(x, i);
        }
    }

    void update_rmq(int idx, int a, int b, int l, int r, int x); /// RMQ
update defined for build
    void build(int root){ /// hash = 397248
        r = root, id = 0, height[r] = 0;
        dfs(r, r);
        build(r, r);
        for (int i = 0; i < n; i++){
            if (up[i] == i) up[i] = parent[i];
        }

        /// Builds RMQ
        clr(lazy);
        for (int i = 0; i < (MAX << 2); i++) tree[i] = NIL;
        for (int i = 0; i < n; i++){
            if (num[i]) update_rmq(1, 1, id, num[i], num[i], nodeval[i]);
        }
    }
    void build(){
        build(0); /// Root set to 0 by default!
    }

    int lca(int a, int b){
        while (up[a] != up[b]){
            if (height[up[a]] > height[up[b]]) a = up[a];
            else b = up[b];
        }

        if (a == b) return a;
        if (num[a] && num[b]){
            if (height[a] < height[b]) return a;
            else return b;
        }
        return up[a];
    }

    void add_edge(int a, int b){
```

```
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    void init(int nodes, int* ar){
        clr(num), n = nodes;
        for (int i = 0; i < MAX; i++) adj[i].clear();
        for (int i = 0; i < n; i++) nodeval[i] = ar[i];
    }

    /************** RMQ functions **************/


    /// Change lazy propagation accordingly
    /// Note lazy and updates set for adding values in node, update if
set/replace operation
    inline void push(int idx, int a, int b){
        int c = (a + b) >> 1, d = c + 1, p = idx << 1, q = p | 1;
        if (lazy[idx]){
            tree[idx] += (lazy[idx] * (b - a + 1)); /// Change lazy
according to operation
            if (a != b) lazy[p] += lazy[idx], lazy[q] += lazy[idx]; ///
Change lazy according to operation
            lazy[idx] = 0;
        }
    }

    /// Change query accordingly
    int query_rmq(int idx, int a, int b, int l, int r){ /// hash = 87775
        int c = (a + b) >> 1, d = c + 1, p = idx << 1, q = p | 1;

        push(idx, a, b);
        if (a == l && b == r) return tree[idx];
        else if (r <= c) return query_rmq(p, a, c, l, r);
        else if (l >= d) return query_rmq(q, d, b, l, r);
        else return OPT(query_rmq(p, a, c, l, c), query_rmq(q, d, b, d,
r));
    }

    /// Change update accordingly
    void update_rmq(int idx, int a, int b, int l, int r, int x){ /// hash
= 487503
        int p = (idx << 1), q = p + 1, c = (a + b) >> 1, d = c + 1;

        if (a == l && b == r) lazy[idx] += x; /// Change lazy according to
operation, change here if set
        push(idx, a, b);
        if (a == l && b == r) return;

        if (r <= c){
            push(q, d, b);
            update_rmq(p, a, c, l, r, x);
        }
        else if (l >= d){
```

```
            push(p, a, c);
            update_rmq(q, d, b, l, r, x);
        }
        else{
            update_rmq(p, a, c, l, c, x);
            update_rmq(q, d, b, d, r, x);
        }

        tree[idx] = OPT(tree[p], tree[q]);
    }

    /************** HLD + RMQ **************/

    /// Sum of all nodes in the path from u to l, l must be an ancestor of
u
    int query_tree(int u, int l){ /// hash = 486879
        int res = NIL;
        while (height[u] > height[l]){
            if (num[u]){
                int v = jump(u);
                if (height[v] <= height[l]) v = down[l];
                res = OPT(res, query_rmq(1, 1, id, num[v], num[u]));
                u = parent[v];
            }
            else res = OPT(nodeval[u], res), u = parent[u];
        }
        return res;
    }

    /// Sum of all nodes in the path from u to v
    int query(int u, int v){
        int l = lca(u, v), res = NIL;
        res = OPT(res, query_tree(u, l));
        res = OPT(res, query_tree(v, l));
        if (!num[l]) res = OPT(nodeval[l], res);
        else res = OPT(query_rmq(1, 1, id, num[l], num[l]), res);
        return res;
    }

    /// Add w to all nodes in the path from u to l, l must be an ancestor
of u
    void update_tree(int u, int l, int w){ /// hash = 423845
        while (height[u] > height[l]){
            if (num[u]){
                int v = jump(u);
                if (height[v] <= height[l]) v = down[l];
                update_rmq(1, 1, id, num[v], num[u], w);
                u = parent[v];
            }
            else nodeval[u] = OPT(nodeval[u], w), u = parent[u]; ///
Change here if set instead of add
        }
    }
```

```
    /// Add w to all nodes in the path from u to v
    void update(int u, int v, int w){
        int l = lca(u, v);
        update_tree(u, l, w);
        update_tree(v, l, w);
        if (!num[l]) nodeval[l] = OPT(nodeval[l], w); /// Change here if
set instead of add
        else update_rmq(1, 1, id, num[l], num[l], w);
    }
}

int n, q, ar[MAX];

int main(){
    int T = 0, t, i, j, k, l, x, u, v, w, res, flag;

    scanf("%d", &t);
    while (t--){
        clr(ar);
        scanf("%d", &n);
        hld::init(n, ar);

        for (i = 1; i < n; i++){
            scanf("%d %d", &u, &v);
            hld::add_edge(u, v);
        }
        hld::build();

        scanf("%d", &q);
        printf("Case #%d:\n", ++T);
        while (q--){
            scanf("%d %d %d", &u, &v, &w);
            hld::update(u, v, w);
        }
        for (i = 0; i < n; i++) printf("%d\n", hld::query(i, i));
    }
    return 0;
}
```

HakmemItem175.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

/// Only for non-negative integers
/// Returns the immediate next number with same count of one bits, -1 on
failure
long long hakmemItem175(long long n){
    if (n == 0) return -1;
    long long x = (n & -n);
```

```
    long long left = (x + n);
    long long right = ((n ^ left) / x) >> 2;
    long long res = (left | right);
    return res;
}

/// Returns the immediate previous number with same count of one bits, -1
on failure
long long lol(long long n){
    if (n == 0 || n == 1) return -1;
    long long res = ~hakmemItem175(~n);
    return (res == 0) ? -1 : res;
}

int main(){

}
```

Hash Table.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define PAD  66667
#define HMOD 1000033
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int id = 1, entry[HMOD + PAD];
unsigned int hashtable[HMOD + PAD];

void insert(unsigned int x){
    int i = x % HMOD;
    while (entry[i] == id && hashtable[i] != x) i++;
    hashtable[i] = x, entry[i] = id;
}

bool find(unsigned int x){
    int i = x % HMOD;
    while (entry[i] == id && hashtable[i] != x) i++;
    return (entry[i] == id);
}

int main(){

}
```

HashMap.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
```

```
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

struct hashmap{
   int t, sz, hmod;
   vector <int> id;
   vector <long long> key, val;

   inline int nextPrime(int n){
       for (int i = n; ;i++){
           for (int j = 2; ;j++){
               if ((j * j) > i) return i;
               if ((i % j) == 0) break;
           }
       }
       return -1;
   }

   void clear(){t++;}

   inline int pos(unsigned long long x){
       int i = x % hmod;
       while (id[i] == t && key[i] != x) i++;
       return i;
   }

   inline void insert(long long x, long long v){
       int i = pos(x);
       if (id[i] != t) sz++;
       key[i] = x, val[i] = v, id[i] = t;
   }

   inline void erase(long long x){
       int i = pos(x);
       if (id[i] == t) key[i] = 0, val[i] = 0, id[i] = 0, sz--;
   }

   inline long long find(long long x){
       int i = pos(x);
       return (id[i] != t) ? -1 : val[i];
   }

   inline bool contains(long long x){
       int i = pos(x);
       return (id[i] == t);
   }

   inline void add(long long x, long long v){
       int i = pos(x);
       (id[i] == t) ? (val[i] += v) : (key[i] = x, val[i] = v, id[i] = t,
sz++);
   }
```

```
    inline int size(){
        return sz;
    }

    hashmap(){}
    hashmap(int m){
        srand(time(0));
        m = (m << 1) - ran(1, m);
        hmod = nextPrime(max(100, m));

        sz = 0, t = 1;
        id.resize(hmod + 0x1FF, 0);
        key.resize(hmod + 0x1FF, 0), val.resize(hmod + 0x1FF, 0);
    }
};

int main(){
}
```

Hashing RMQ.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define LOG 18
#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int n, lim;
char str[MAX];
unsigned long long P[MAX], hash[MAX][LOG];
const unsigned long long base = 1968647011ULL;

void Generate(bool gen){
    int i, j, l;
    lim = 32 - __builtin_clz(n);

    if (gen){
        P[0] = 1ULL;
        for (i = 1; i < MAX; i++) P[i] = (P[i - 1] * base);
    }

    for (l = 0; l < lim; l++){
        int len = (1 << l);
        for (i = 0; (i + len) <= n; i++){
            if (l == 0) hash[i][l] = str[i];
            else{
                int d = 1 << (l - 1);
                hash[i][l] = (P[d] * hash[i][l - 1]) + hash[i + d][l - 1];
            }
        }
```

```
    }
}

unsigned long long GetHash(int i, int j){
    int d, l = (j - i + 1);
    unsigned long long res = 0;

    for (d = lim - 1; d >= 0; d--){
        if (l & (1 << d)){
            res = (res * P[1 << d]) + hash[i][d];
            i += (1 << d);
        }
    }

    return res;
}

int main(){
    scanf("%s", str);
    n = strlen(str);
    Generate();
    return 0;
}
```

Heavy Light + RMQ.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 30010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

vector <int> ar[MAX];
int t, n, m, val[MAX], tree[MAX << 2];
int h, root, subtree[MAX], depth[MAX], parent[MAX], head[MAX], tail[MAX],
chain[MAX], num[MAX];

void dfs(int i, int p){
    parent[i] = p;
    subtree[i] = 1;
    if (i != p) depth[i] = depth[p] + 1;

    int len = ar[i].size();
    for (int j = 0; j < len; j++){
        int x = ar[i][j];
        if (x != p){
            dfs(x, i);
            subtree[i] += subtree[x];
        }
    }
}
```

```
void hld(int i, int p){
   int heavy = -1;
   int len = ar[i].size();

   chain[i] = i;
   if (head[i] != -1) chain[i] = chain[p];

   for (int j = 0; j < len; j++){
       int x = ar[i][j];
       if ((x != p) && ((subtree[x] << 1) >= subtree[i])){
           heavy = x;
           head[x] = i;
           tail[i] = x;
       }
   }


   if (heavy != -1){
       num[heavy] = ++h;
       hld(heavy, i);
   }

   for (int j = 0; j < len; j++){
       int x = ar[i][j];
       if ((x != p) && (x != heavy)) hld(x, i);
   }
}

void HeavyLight(){
   h = 0;
   root = 0;
   memset(head, -1, sizeof(head));
   memset(tail, -1, sizeof(tail));

   depth[root] = 0;
   dfs(root, root);
   hld(root, root);
   for (int i = 0; i < n; i++){
       if (i == root) chain[i] = 0;
       else if (chain[i] == i) chain[i] = parent[i];
   }
}

int lca(int a, int b){
   while (chain[a] != chain[b]){
       if (depth[chain[a]] > depth[chain[b]]) a = chain[a];
       else b = chain[b];
   }

   if (a == b) return a;
   if (head[a] != -1 && head[b] != -1){
       if (depth[a] < depth[b]) return a;
       else return b;
```

```
    }

    return chain[a];
}

int query(int idx){
    int res = 0;
    while (idx > 0){
        res += tree[idx];
        idx ^= (idx & -idx);
    }

    return res;
}

void update(int idx, int value){
    while (idx <= h){
        tree[idx] += value;
        idx += (idx & -idx);
    }
}

void RMQ(){
    clr(tree);
    for (int i = 0; i < n; i++){
        if (head[i] != -1){
            update(num[i], val[i]);
        }
    }
}

void update_node(int u, int v){
    if (head[u] != -1){
        update(num[u], -val[u]);
        val[u] = v;
        update(num[u], val[u]);
    }
    else val[u] = v;
}

int F(int v){
    int res = 0;

    for (; ;){
        if (head[v] != -1){
            int x = chain[v];
            int d = depth[v] - depth[x];
            int idx = num[v];
            int r = query(idx) - query(idx - d);
            res += r;
            v = x;
        }
        else{
            res += val[v];
```

```
            if (!v) break;
            v = parent[v];
        }
    }
    return res;
}

int Solve(int a, int b){
    int l = lca(a, b);
    int x = F(a) + F(b);
    int y = F(l) << 1;
    int res = (x - y) + val[l];
    return res;
}

int main(){
    int T = 0, i, j, q, a, b;

    scanf("%d", &t);
    while (t--){
        scanf("%d", &n);
        for (i = 0; i <= n; i++) ar[i].clear();

        for (i = 0; i < n; i++) scanf("%d", &val[i]);
        for (i = 1; i < n; i++){
            scanf("%d %d", &a, &b);
            ar[a].push_back(b);
            ar[b].push_back(a);
        }

        HeavyLight();
        RMQ();

        printf("Case %d:\n", ++T);
        scanf("%d", &m);
        while (m--){
            scanf("%d %d %d", &q, &a, &b);
            if (!q) printf("%d\n", Solve(a, b));
            else update_node(a, b);
        }
    }
    return 0;
}
```

Histogram Area.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 2010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
```

```
int n, ar[MAX];

/*** Largest area in a histogram ***/
/*** Be careful, long long might be required ***/

int histogram(int n, int* ar){
   int i = 0, j, a, x, top = 0, res = 0;

   stack[0] = -1;
   while (i < n){
       if (!top || (ar[stack[top]] <= ar[i]) ) stack[++top] = i++;
       else{
           x = stack[top--];
           a = ar[x] * (i - stack[top] - 1);
           if (a > res) res = a;
       }
   }

   while (top){
       x = stack[top--];
       a = ar[x] * (i - stack[top] - 1);
       if (a > res) res = a;
   }

   return res;
}

int main(){

}
```

Hopcroft Karp.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 1010
#define clr(ar) (memset(ar, 0, sizeof(ar)))
#define read() freopen("lol.txt", "r", stdin)

bool bfs_vis[MAX];
int t, n, m, ar[MAX][MAX], len[MAX], left[MAX], right[MAX], Q[MAX],
dis[MAX], parent[MAX];

bool dfs(int i){
   int j, x;

   for (j = 0; j < len[i]; j++){
       x = ar[i][j];
       if (left[x] == -1 || (parent[left[x]] == i)){
           if (left[x] == -1 || dfs(left[x])){
               left[x] = i;
               right[i] = x;
```

```
                return true;
            }
        }
    }
    return false;
}

bool bfs(){
    clr(bfs_vis);
    int i, j, x, d, f = 0, l = 0;

    for (i = 0; i < n; i++){
        if (right[i] == -1){
            Q[l++] = i;
            dis[i] = 0;
            bfs_vis[i] = true;
        }
    }

    while (f < l){
        i = Q[f++];

        for (j = 0; j < len[i]; j++){
            x = ar[i][j];
            d = left[x];
            if (d == -1) return true;

            else if (!bfs_vis[d]){
                Q[l++] = d;
                parent[d] = i;
                bfs_vis[d] = true;
                dis[d] = dis[i] + 1;
            }
        }
    }
    return false;
}

int hopcroft_karp(){
    int i, j, counter = 0;
    memset(left, -1, sizeof(left));
    memset(right, -1, sizeof(right));

    while (bfs()){
        for (i = 0; i < n; i++){
            if (right[i] == -1 && dfs(i)) counter++;
        }
    }
    return counter;
}

int main(){
    int i, a, b;
```

```
    while (scanf("%d %d", &n, &m) != EOF){
        clr(len);
        for (i = 0; i < m; i++){
            scanf("%d %d", &a, &b);
            ar[a][len[a]++] = b;
        }
    }
    return 0;
}
```

Hopcroft Karp.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Hopcroft karp in O(m * sqrt(n))
namespace hc{ /// hash = 393558
    bool visited[MAX];
    vector <int> adj[MAX];
    int n, L[MAX], R[MAX], Q[MAX], len[MAX], dis[MAX], parent[MAX];

    inline void init(int nodes){ /// Number of vertices in the left set,
or max(left_set, right_set)
        n = nodes, clr(len);
        for (int i = 0; i < MAX; i++) adj[i].clear();
    }

    inline void add_edge(int u, int v){ /// 0 based index
        len[u]++;
        adj[u].push_back(v);
    }

    bool dfs(int i){
        for (int j = 0; j < len[i]; j++){
            int x = adj[i][j];
            if (L[x] == -1 || (parent[L[x]] == i)){
                if (L[x] == -1 || dfs(L[x])){
                    L[x] = i, R[i] = x;
                    return true;
                }
            }
        }
        return false;
    }

    bool bfs(){
        clr(visited);
```

```
        int i, j, x, d, f = 0, l = 0;

        for (i = 0; i < n; i++){
            if (R[i] == -1){
                visited[i] = true;
                Q[l++] = i, dis[i] = 0;
            }
        }

        while (f < l){
            i = Q[f++];
            for (j = 0; j < len[i]; j++){
                x = adj[i][j], d = L[x];
                if (d == -1) return true;

                else if (!visited[d]){
                    Q[l++] = d;
                    parent[d] = i, visited[d] = true, dis[d] = dis[i] + 1;
                }
            }
        }
        return false;
    }

    int hopcroft_karp(){
        int res = 0;
        memset(L, -1, sizeof(L));
        memset(R, -1, sizeof(R));

        while (bfs()){
            for (int i = 0; i < n; i++){
                if (R[i] == -1 && dfs(i)) res++;
            }
        }
        return res;
    }
}

int main(){
    int n, m, r, i, j, a, b;

    scanf("%d %d %d", &n, &m, &r);
    hc::init(max(n, m));
    while (r--){
        scanf("%d %d", &a, &b);
        hc::add_edge(--a, --b);
    }

    printf("%d\n", hc::hopcroft_karp());
    return 0;
}
```

Hungarian Algorithm.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 666
#define inf (~0U >> 1)
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

bool used[MAX];
int n, m, ar[MAX][MAX];
int U[MAX], V[MAX], P[MAX], way[MAX], minv[MAX], match[MAX];

/*** Hungarian Algorithm for weighted bipartite matching ***/
/*** n = number of rows,m = number of columns, in 1-base ***/
/*** match[i] contains the column to which it is matched ***/
/*** n = Note that m should be >= n, Just fill the whole matrix with 0 if
this is not true ***/

int hungarian(int n, int m, int ar[MAX][MAX]){
   if (n > m) m = n;
   clr(way), clr(U), clr(V), clr(P);
   int i, j, i0, i1, j0, j1, cur, delta;

   for (i = 1; i <= n; i++){
       P[0] = i, j0 = 0;
       for (j = 0; j <= m; j++) minv[j] = inf, used[j] = false;

       do{
           used[j0] = true;
           i0 = P[j0], j1 = 0, delta = inf;

           for (j = 1; j <= m; j++){
               if (!used[j]){
                   cur = ar[i0][j] - U[i0] - V[j];
                   if (cur < minv[j]){
                       minv[j] = cur;
                       way[j] = j0;
                   }
                   if (minv[j] < delta){
                       delta = minv[j];
                       j1 = j;
                   }
               }
           }

           for (j = 0; j <= m; j++){
               if (used[j]){
                   U[P[j]] += delta;
                   V[j] -= delta;
               }
               else minv[j] -= delta;
           }
           j0 = j1;
```

```
        } while (P[j0] != 0);

        do{
            j1 = way[j0];
            P[j0] = P[j1];
            j0 = j1;
        } while (j0 != 0);
    }
    for (j = 1; j <= m; j++) match[P[j]] = j;

    return -V[0];
}

int main(){
}
```

Hungarian Algorithm.cpp

```
-------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 666
#define MAXIMIZE +1
#define MINIMIZE -1

#define inf (~0U >> 1)
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

namespace wm{ /// hash = 581023
   bool visited[MAX];
   int U[MAX], V[MAX], P[MAX], way[MAX], minv[MAX], match[MAX],
ar[MAX][MAX];

   /// n = number of row and m = number of columns in 1 based, flag =
MAXIMIZE or MINIMIZE
   /// match[i] contains the column to which row i is matched
   int hungarian(int n, int m, int mat[MAX][MAX], int flag){
       clr(U), clr(V), clr(P), clr(ar), clr(way);

       for (int i = 1; i <= n; i++){
           for (int j = 1; j <= m; j++){
               ar[i][j] = mat[i][j];
               if (flag == MAXIMIZE) ar[i][j] = -ar[i][j];
           }
       }
       if (n > m) m = n;

       int i, j, a, b, c, d, r, w;
       for (i = 1; i <= n; i++){
           P[0] = i, b = 0;
```

```
            for (j = 0; j <= m; j++) minv[j] = inf, visited[j] = false;

            do{
                visited[b] = true;
                a = P[b], d = 0, w = inf;

                for (j = 1; j <= m; j++){
                    if (!visited[j]){
                        r = ar[a][j] - U[a] - V[j];
                        if (r < minv[j]) minv[j] = r, way[j] = b;
                        if (minv[j] < w) w = minv[j], d = j;
                    }
                }

                for (j = 0; j <= m; j++){
                    if (visited[j]) U[P[j]] += w, V[j] -= w;
                    else minv[j] -= w;
                }
                b = d;
            } while (P[b] != 0);

            do{
                d = way[b];
                P[b] = P[d], b = d;
            } while (b != 0);
        }
        for (j = 1; j <= m; j++) match[P[j]] = j;

        return (flag == MINIMIZE) ? -V[0] : V[0];
    }
}

int main(){
}
```

Hunt-Szymanski.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 50010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

int ar[MAX];
char A[MAX], B[MAX];

/// Hunt-Szymanski Algorithm for LCS
/// O(R + N) log N, R = numbered of ordered pairs of positions where the
two strings match (worst case, R = N^2)

int lcs(char* A, char* B){ /// hash = 935751
```

```
   vector <int> adj[256];
   int i, j, l = 0, n = strlen(A), m = strlen(B);
   for (i = 0; i < m; i++) adj[B[i]].push_back(i);

   ar[l++] = -1;
   for (i = 0; i < n; i++){
       for (j = adj[A[i]].size() - 1; j >= 0; j--){
           int x = adj[A[i]][j];
           if (x > ar[l - 1]) ar[l++] = x;
           else ar[lower_bound(ar, ar + l, x) - ar] = x;
       }
   }
   return l - 1;
}

int main(){
   int i, j, k, n, m;
   n = MAX - 10, m = MAX - 10;
   for (i = 0; i < n; i++) A[i] = (rand() % 26) + 'A';
   for (i = 0; i < m; i++) B[i] = (rand() % 26) + 'A';
   A[n] = B[m] = 0;

   clock_t start = clock();
   printf("%d\n", lcs(A, B));

   printf("%0.6f\n", (clock() - start) / (1.0 * CLOCKS_PER_SEC));
   return 0;
}
```

```
IDAStar.c
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int ida(int g, int lim, int l, int last, int idx){ /// last = last taken
move, don't go there!
   int h = heuristic(l);
   if (!h){ /// Goal reached
       sequence[idx] = 0;
       puts(sequence);
       return g;
   }

   int f = g + h;
   if (f > lim) return f;
   int i, j, res = inf;
   for (i = 0; i < 12; i++){
       if (dis[l][i] == 1 && i != last){
           sequence[idx] = str[i];
           swap(str[l], str[i]);
```

```
            int x = ida(g + 1, lim, i, l, idx + 1);
            if (x < res) res = x; /// Update next limit in iterative
deepening
            swap(str[l], str[i]);
            if (res <= lim) return res; /// Since iterative deepening,
return if solution found
        }
    }
    return res;
}

int Solve(int l){
    int lim = heuristic(l);
    for (; ;){
        int nlim = ida(0, lim, l, l, 0);
        if (nlim <= lim) return nlim;
        else lim = nlim;
    }
    return -1;
}
```

Integral Determinant LL.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 1010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

const long long MOD = 4517409488245517117LL;
const long double OP = (long double)1 / 4517409488245517117LL;

long long mul(long long a, long long b){
  long double res = a;
  res *= b;
  long long c = (long long)(res * OP);
  a *= b;
  a -= c * MOD;
  if (a >= MOD) a -= MOD;
  if (a < 0) a += MOD;
  return a;
}

long long expo(long long x, long long n){
    long long res = 1;

    while (n){
        if (n & 1) res = mul(res, x);
        x = mul(x, x);
        n >>= 1;
    }
```

```
    return res;
}

int gauss(int n, long long ar[MAX][MAX]){
    long long x, y;
    int i, j, k, l, p, counter = 0;

    for (i = 0; i < n; i++){
        for (p = i, j = i + 1; j < n && !ar[p][i]; j++){
            p = j;
        }
        if (!ar[p][i]) return -1;

        for (j = i; j < n; j++){
            x = ar[p][j], ar[p][j] = ar[i][j], ar[i][j] = x;
        }

        if (p != i) counter++;
        for (j = i + 1; j < n; j++){
            x = expo(ar[i][i], MOD - 2);
            x = mul(x, MOD - ar[j][i]);

            for (k = i; k < n; k++){
                ar[j][k] = ar[j][k] + mul(x, ar[i][k]);
                if (ar[j][k] >= MOD) ar[j][k] -= MOD;
            }
        }
    }
    return counter;
}

/// Finds the determinant of a square matrix
/// Returns 0 if the matrix is singular or degenerate (hence no
determinant exists)
/// Absolute value of final answer should be < MOD / 2

long long determinant(int n, long long ar[MAX][MAX]){
    int i, j, free;
    long long res = 1;

    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            if (ar[i][j] < 0) ar[i][j] += MOD;
        }
    }

    free = gauss(n, ar);
    if (free == -1) return 0; /// Determinant is 0 so matrix is not
invertible, singular or degenerate matrix

    for (i = 0; i < n; i++) res = mul(res, ar[i][i]);
    if (free & 1) res = MOD - res;
    if ((MOD - res) < res) res -= MOD; /// Determinant can be negative so
if determinant is more close to MOD than 0, make it negative
```

```
   return res;
}

int n;
long long ar[MAX][MAX];

int main(){
   int t, i, j, k, l;

   while (scanf("%d", &n) != EOF){
       if (n == 0) break;

       for (i = 0; i < n; i++){
           for (j = 0; j < n; j++){
               scanf("%lld", &ar[i][j]);
           }
       }

       printf("%lld\n", determinant(n, ar));
   }
   return 0;
}

Integral Determinant.c
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 1010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int expo(int a, int b, int MOD){
   int res = 1;

   while (b){
       if (b & 1) res = (long long)res * a % MOD;
       a = (long long)a * a % MOD;
       b >>= 1;
   }
   return res;
}

int gauss(int n, int ar[MAX][MAX], int MOD){ /// Gaussian elimination in
field MOD (MOD should be a prime)
   int i, j, k, l, p, counter = 0;

   for (i = 0; i < n; i++){
       for (p = i, j = i + 1; j < n && !ar[p][i]; j++){
           p = j;
       }
       if (!ar[p][i]) return -1;
```

```
        for (j = i; j < n; j++){
            k = ar[p][j], ar[p][j] = ar[i][j], ar[i][j] = k;
        }

        if (p != i) counter++;
        for (j = i + 1; j < n; j++){
            long long x = (long long)expo(ar[i][i], MOD - 2, MOD) * (MOD -
ar[j][i]) % MOD;
            for (k = i; k < n; k++){
                ar[j][k] = (x * ar[i][k] + ar[j][k]) % MOD;
            }
        }
    }
    return counter;
}

/// Finds the determinant of a square matrix
/// Returns 0 if the matrix is singular or degenerate (hence no
determinant exists)

int determinant(int n, int ar[MAX][MAX], int MOD){
    long long res = 1;
    int i, j, k, free;
    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            if (ar[i][j] < 0) ar[i][j] += MOD;
        }
    }

    free = gauss(n, ar, MOD);
    if (free == -1) return 0; /// Determinant is 0 so matrix is not
invertible, singular or degenerate matrix

    for (i = 0; i < n; i++) res = (res * ar[i][i]) % MOD;
    if (free & 1) res = MOD - res;
    if ((MOD - res) < res) res -= MOD; /// Determinant can be negative so
if determinant is more close to MOD than 0, make it negative

    return res;
}

int n, ar[MAX][MAX];

int main(){
    int t, i, j, k, l;
    const int MOD = 1000000007;

    while (scanf("%d", &n) != EOF){
        if (n == 0) break;

        for (i = 0; i < n; i++){
            for (j = 0; j < n; j++){
                scanf("%d", &ar[i][j]);
```

```
            }
        }

        printf("%d\n", determinant(n, ar, MOD));
    }
    return 0;
}
```

Inverse Factorial (Linear).c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 100010
#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int fact[MAX], inv[MAX];

int expo(int a, int b){
    int res = 1;

    while (b){
        if (b & 1) res = (long long)res * a % MOD;
        a = (long long)a * a % MOD;
        b >>= 1;
    }
    return res;
}

void Generate(){
    int i, x;
    for (fact[0] = 1, i = 1; i < MAX; i++) fact[i] = ((long long)i *
fact[i - 1]) % MOD;

    inv[MAX - 1] = expo(fact[MAX - 1], MOD - 2);
    for (i = MAX - 2; i >= 0; i--) inv[i] = ((long long)inv[i + 1] * (i +
1)) % MOD;
}

int main(){
    Generate();
    printf("%d\n", inv[35]);
    printf("%d\n", expo(fact[35], MOD - 2));
    return 0;
}
```

Inverse Modulo 1 to N (Linear).c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
```

```
#define MAX 100010
#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int fact[MAX], inv[MAX];

int expo(int a, int b){
   int res = 1;

   while (b){
       if (b & 1) res = (long long)res * a % MOD;
       a = (long long)a * a % MOD;
       b >>= 1;
   }
   return res;
}

void Generate(){
   int i, x;
   for (fact[0] = 1, i = 1; i < MAX; i++) fact[i] = ((long long)i *
fact[i - 1]) % MOD;

   /// inv[i] = Inverse modulo of fact[i]
   inv[MAX - 1] = expo(fact[MAX - 1], MOD - 2);
   for (i = MAX - 2; i >= 0; i--) inv[i] = ((long long)inv[i + 1] * (i +
1)) % MOD;

   /// Inverse modulo of numbers 1 to MAX in linear time below
   inv[1] = 1;
      for (i = 2; i < MAX; i++){
       inv[i] = MOD - ((MOD / i) * (long long)inv[MOD % i]) % MOD;
       if (inv[i] < 0) inv[i] += MOD;
      }
}

int main(){
   Generate();
   printf("%d\n", inv[35]);
   printf("%d\n", expo(fact[35], MOD - 2));
   return 0;
}

Johnson_s Algorithm.cpp
--------------------------------------------------
#include <stdio.h>
#include <bits/stdtr1c++.h>

#define MAX 2525
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define valid(i, j) ((i) >= 1 && (i) <= n && (j) >= 1 && (j) <= m)
```

```
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Johnson's algorithm for all pair shortest paths in sparse graphs
/// Complexity: O(N * M) + O(N * M * log(N))

const long long INF = (1LL << 60) - 666;

struct edge{
   int u, v;
   long long w;
   edge(){}
   edge(int u, int v, long long w) : u(u), v(v), w(w){}

   void print(){
       cout << "edge " << u << " " << v << " " << w << endl;
   }
};

bool bellman_ford(int n, int src, vector <struct edge> E, vector <long
long>& dis){
   dis[src] = 0;
   for (int i = 0; i <= n; i++){
       int flag = 0;
       for (auto e: E){
           if ((dis[e.u] + e.w) < dis[e.v]){
               flag = 1;
               dis[e.v] = dis[e.u] + e.w;
           }
       }
       if (flag == 0) return true;
   }
   return false;
}

vector <long long> dijkstra(int n, int src, vector <struct edge> E,
vector <long long> potential){
   set<pair<long long, int> > S;
   vector <long long> dis(n + 1, INF);
   vector <long long> temp(n + 1, INF);
   vector <pair<int, long long> > adj[n + 1];

   dis[src] = temp[src] = 0;
   S.insert(make_pair(temp[src], src));
   for (auto e: E){
       adj[e.u].push_back(make_pair(e.v, e.w));
   }

   while (!S.empty()){
       pair<long long, int> cur = *(S.begin());
       S.erase(cur);

       int u = cur.second;
```

```
       for (int i = 0; i < adj[u].size(); i++){
           int v = adj[u][i].first;
           long long w = adj[u][i].second;

           if ((temp[u] + w) < temp[v]){
               S.erase(make_pair(temp[v], v));
               temp[v] = temp[u] + w;
               dis[v] = dis[u] + w;
               S.insert(make_pair(temp[v], v));
           }
       }
   }
   return dis;
}

void johnson(int n, long long ar[MAX][MAX], vector <struct edge> E){
   vector <long long> potential(n + 1, INF);
   for (int i = 1; i <= n; i++) E.push_back(edge(0, i, 0));

   assert(bellman_ford(n, 0, E, potential));
   for (int i = 1; i <= n; i++) E.pop_back();

   for (int i = 1; i <= n; i++){
       vector <long long> dis = dijkstra(n, i, E, potential);
       for (int j = 1; j <= n; j++){
           ar[i][j] = dis[j];
       }
   }
}

long long ar[MAX][MAX];

int main(){
   vector <struct edge> E;
   E.push_back(edge(1, 2, 2));
   E.push_back(edge(2, 3, -15));
   E.push_back(edge(1, 3, -10));

   int n = 3;
   johnson(n, ar, E);
   for (int i = 1; i <= n; i++){
       for (int j = 1; j <= n; j++){
           printf("%d %d = %lld\n", i, j, ar[i][j]);
       }
   }
   return 0;
}

Josephus Problem.c
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
```

```
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

/// Josephus problem, n people numbered from 1 to n stand in a circle.
/// Counting starts from 1 and every k'th people dies
/// Returns the position of the m'th killed people
/// For example if n = 10 and k = 3, then the people killed are 3, 6, 9,
2, 7, 1, 8, 5, 10, 4 respectively

/// O(n)
int josephus(int n, int k, int m){
   int i;
   for (m = n - m, i = m + 1; i <= n; i++){
       m += k;
       if (m >= i) m %= i;
   }
   return m + 1;
}

/// O(k log(n))
long long josephus2(long long n, long long k, long long m){ /// hash =
583016
   m = n - m;
   if (k <= 1) return n - m;

   long long i = m;
   while (i < n){
       long long r = (i - m + k - 2) / (k - 1);
       if ((i + r) > n) r = n - i;
       else if (!r) r = 1;
       i += r;
       m = (m + (r * k)) % i;
   }
   return m + 1;
}

int main(){
   int n, k, m;
   printf("%d\n", josephus(10, 1, 2));
   printf("%d\n", josephus(10, 1, 10));
}
```

Judge Data File.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;
using namespace tr1;
```

```
char ch, str[MAX], str2[MAX];

int main(){
   FILE *test1;
   test1 = fopen("test1.txt", "r");
   FILE *test2;
   test2 = fopen("test2.txt", "r");

   int counter = 0;
   for (int i = 1; ;i++){
       fscanf(test1, "%[^\n]", str);
       fscanf(test1, "%c", &ch);
       fscanf(test2, "%[^\n]", str2);
       fscanf(test2, "%c", &ch);
       if (feof(test1) || feof(test2)){
           if (!counter && feof(test1) && feof(test2)) puts("Accepted");
           else{
               puts("Wrong Answer");
               printf("%d Mistakes\n", counter);
           }
           break;
       }

       if (strcmp(str, str2) != 0){
           counter++;
           printf("Line %d\n", i);
       }
   }

   fflush(test1);
   fflush(test2);
   return 0;
}
```

K_TH Number.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

pair<int, int> val[MAX];
int n, ar[MAX], pos[MAX];
vector<int> tree[MAX << 2];

void merge_sort_tree(int idx, int a, int b, int* ar){ /// hash = 974987
   int p = idx << 1, q = p | 1, c = (a + b) >> 1, d = c + 1;
   int i = 0, j = 0, k = 0, u = c - a + 1, v = b - d + 1, len = b - a +
1;
```

```
    tree[idx].resize(len, 0);
    if (a == b){
        tree[idx][0] = ar[a];
        return;
    }

    merge_sort_tree(p, a, c, ar);
    merge_sort_tree(q, d, b, ar);
    while (len--){
        if (i == u) tree[idx][k++] = tree[q][j++];
        else if (j == v) tree[idx][k++] = tree[p][i++];
        else if (tree[p][i] < tree[q][j]) tree[idx][k++] = tree[p][i++];
        else tree[idx][k++] = tree[q][j++];
    }
}

void build(){
    for (int i = 1; i <= n; i++) val[i] = make_pair(ar[i], i);
    sort(val + 1, val + 1 + n);
    for (int i = 1; i <= n; i++) pos[i] = val[i].second;
    merge_sort_tree(1, 1, n, pos);
}

int query(int l, int r, int k){ /// hash = 939184
    int m, c, a = 1, b = n, idx = 1;

    while (a != b){
        m = (a + b) >> 1, idx <<= 1;
        c = upper_bound(tree[idx].begin(), tree[idx].end(), r) -
lower_bound(tree[idx].begin(), tree[idx].end(), l);
        if (c >= k) b = m;
        else k -= c, idx |= 1, a = ++m;
    }
    return val[a].first;
}

int main(){
    int i, j, k, x, q, l, r;

    while (scanf("%d %d", &n, &q) != EOF){
        for (i = 1; i <= n; i++) scanf("%d", &ar[i]);
        build();

        while (q--){
            scanf("%d %d %d", &l, &r, &k);
            printf("%d\n", query(l, r, k));
        }
    }
    return 0;
}
```

Karatsuba LL.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 131072 /// Must be a power of 2
#define MOD 1000000007

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

int ptr = 0;
long long temp[128];
long long buffer[MAX * 6];
const long long INF = 8000000000000000000LL;

void karatsuba(int n, long long *a, long long *b, long long *res){ /// n
is a power of 2
   int i, j, s;
      if (n < 33){ /// Reduce recursive calls by setting a threshold
       for (i = 0; i < (n + n); i++) temp[i] = 0;
       for (i = 0; i < n; i++){
            if (a[i]){
                 for (j = 0; j < n; j++){
                      temp[i + j] += (a[i] * b[j]);
                      if (temp[i + j] > INF) temp[i + j] %= MOD;
                 }
            }
       }
            for (i = 0; i < (n + n); i++) res[i] = temp[i] % MOD;
            return;
      }

      s = n >> 1;
      karatsuba(s, a, b, res);
      karatsuba(s, a + s, b + s, res + n);
      long long *x = buffer + ptr, *y = buffer + ptr + s, *z = buffer +
ptr + s + s;

      ptr += (s + s + n);
      for (i = 0; i < s; i++){
       x[i] = a[i] + a[i + s], y[i] = b[i] + b[i + s];
       if (x[i] >= MOD) x[i] -= MOD;
       if (y[i] >= MOD) y[i] -= MOD;
      }

      karatsuba(s, x, y, z);
      for (i = 0; i < n; i++) z[i] -= (res[i] + res[i + n] - MOD);
      for (i = 0; i < n; i++) res[i + s] = (res[i + s] + z[i] + MOD) %
MOD;
      ptr -= (s + s + n);
}
```

```
/// multiplies two polynomial a(degree n) and b(degree m) and returns the
result modulo MOD in a
/// returns the degree of the multiplied polynomial
/// note that a and b are changed in the process
int mul(int n, long long *a, int m, long long *b){
   int i, r, c = (n < m ? n : m), d = (n > m ? n : m), *res = buffer +
ptr;
   r = 1 << (32 - __builtin_clz(d) - (__builtin_popcount(d) == 1));
   for (i = d; i < r; i++) a[i] = b[i] = 0;
   for (i = c; i < d && n < m; i++) a[i] = 0;
   for (i = c; i < d && m < n; i++) b[i] = 0;

   ptr += (r << 1), karatsuba(r, a, b, res), ptr -= (r << 1);
   for (i = 0; i < (r << 1); i++) a[i] = res[i];
   return (n + m - 1);
}

long long a[MAX], b[MAX];

int main(){
   int i, j, k, n = MAX - 10;
   for (i = 0; i < n; i++) a[i] = ran(1, 1000000000);
   for (i = 0; i < n; i++) b[i] = ran(1, 991929183);
   clock_t start = clock();
   mul(n, a, n, b);
   dbg(a[n / 2]);
   for (i = 0; i < (n << 1); i++){
       if (a[i] < 0) puts("YO");
   }
   printf("%0.5f\n", (clock() - start) / (1.0 * CLOCKS_PER_SEC));
   return 0;
}
```

Karatsuba Unrolled.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 131072 /// Must be a power of 2
#define MOD 1000000007

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

int ptr = 0, buffer[MAX * 6];
unsigned long long temp[128];
const unsigned long long INF = 16111345678373523861LL;

void karatsuba(int n, int *a, int *b, int *res){ /// n is a power of 2
   int i, j, h;
   /// MOD * MOD * n must fit in unsigned long long!
```

```
    if (n == 16){ /// Loop unrolling, to reduce recursive calls
        for (i = 0; i < (n + n); i++) temp[i] = 0;
        for (i = 0; i < n; i++){
            temp[i + 0] += ((long long)a[i] * b[0]), temp[i + 1] += ((long
long)a[i] * b[1]), temp[i + 2] += ((long long)a[i] * b[2]), temp[i + 3]
+= ((long long)a[i] * b[3]);
            temp[i + 4] += ((long long)a[i] * b[4]), temp[i + 5] += ((long
long)a[i] * b[5]), temp[i + 6] += ((long long)a[i] * b[6]), temp[i + 7]
+= ((long long)a[i] * b[7]);
            temp[i + 8] += ((long long)a[i] * b[8]), temp[i + 9] += ((long
long)a[i] * b[9]), temp[i + 10] += ((long long)a[i] * b[10]), temp[i +
11] += ((long long)a[i] * b[11]);
            temp[i + 12] += ((long long)a[i] * b[12]), temp[i + 13] +=
((long long)a[i] * b[13]), temp[i + 14] += ((long long)a[i] * b[14]),
temp[i + 15] += ((long long)a[i] * b[15]);
        }
            for (i = 0; i < (n + n); i++) res[i] = temp[i] % MOD;
            return;
      }

      if (n < 17){ /// Reduce recursive calls by setting a threshold
        for (i = 0; i < (n + n); i++) temp[i] = 0;
        for (i = 0; i < n; i++){
            for (j = 0; j < n; j++){
                temp[i + j] += ((long long)a[i] * b[j]);
            }
        }
            for (i = 0; i < (n + n); i++) res[i] = temp[i] % MOD;
            return;
      }

      h = n >> 1;
      karatsuba(h, a, b, res);
      karatsuba(h, a + h, b + h, res + n);
      int *x = buffer + ptr, *y = buffer + ptr + h, *z = buffer + ptr + h
+ h;

      ptr += (h + h + n);
      for (i = 0; i < h; i++){ /// Loop unrolling
        x[i] = a[i] + a[i + h], y[i] = b[i] + b[i + h];
        if (x[i] >= MOD) x[i] -= MOD;
        if (y[i] >= MOD) y[i] -= MOD;

        i++;
        x[i] = a[i] + a[i + h], y[i] = b[i] + b[i + h];
        if (x[i] >= MOD) x[i] -= MOD;
        if (y[i] >= MOD) y[i] -= MOD;
      }

      karatsuba(h, x, y, z);
      for (i = 0; i < n; i += 2){ /// Loop unrolling
        z[i] -= (res[i] + res[i + n]);
        z[i + 1] -= (res[i + 1] + res[i + n + 1]);
```

```
      }

      for (i = 0; i < n; i++){ /// Loop unrolling
        res[i + h] = (res[i + h] + z[i]) % MOD;
        if (res[i + h] < 0) res[i + h] += MOD;
        i++;
        res[i + h] = (res[i + h] + z[i]) % MOD;
        if (res[i + h] < 0) res[i + h] += MOD;
      }
      ptr -= (h + h + n);
}

/// multiplies two polynomial a(degree n) and b(degree m) and returns the
result modulo MOD in a
/// returns the degree of the multiplied polynomial
/// note that a and b are changed in the process
int mul(int n, int *a, int m, int *b){
   int i, r, c = (n < m ? n : m), d = (n > m ? n : m), *res = buffer +
ptr;
   r = 1 << (32 - __builtin_clz(d) - (__builtin_popcount(d) == 1));
   for (i = d; i < r; i++) a[i] = b[i] = 0;
   for (i = c; i < d && n < m; i++) a[i] = 0;
   for (i = c; i < d && m < n; i++) b[i] = 0;

   ptr += (r << 1), karatsuba(r, a, b, res), ptr -= (r << 1);
   for (i = 0; i < (r << 1); i++) a[i] = res[i];
   return (n + m - 1);
}

int a[MAX], b[MAX];

int main(){
   int i, j, k, n = MAX - 10;
   for (i = 0; i < n; i++) a[i] = ran(1, 1000000000);
   for (i = 0; i < n; i++) b[i] = ran(1, 991929183);
   clock_t start = clock();
   mul(n, a, n, b);
   dbg(a[n / 2]);
   for (i = 0; i < (n << 1); i++){
       if (a[i] < 0) puts("YO");
   }
   printf("%0.5f\n", (clock() - start) / (1.0 * CLOCKS_PER_SEC));
   return 0;
}
```

Karatsuba.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 131072 /// Must be a power of 2
#define MOD 1000000007

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
```

```
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

unsigned long long temp[128];
int ptr = 0, buffer[MAX * 6];

/// n is a power of 2
void karatsuba(int n, int *a, int *b, int *res){ /// hash = 829512
   int i, j, h;
      if (n < 17){ /// Reduce recursive calls by setting a threshold
        for (i = 0; i < (n + n); i++) temp[i] = 0;
        for (i = 0; i < n; i++){
            if (a[i]){
                for (j = 0; j < n; j++){
                    temp[i + j] += ((long long)a[i] * b[j]);
                }
            }
        }
            for (i = 0; i < (n + n); i++) res[i] = temp[i] % MOD;
            return;
      }

      h = n >> 1;
      karatsuba(h, a, b, res);
      karatsuba(h, a + h, b + h, res + n);
      int *x = buffer + ptr, *y = buffer + ptr + h, *z = buffer + ptr + h
+ h;

      ptr += (h + h + n);
      for (i = 0; i < h; i++){
        x[i] = a[i] + a[i + h], y[i] = b[i] + b[i + h];
        if (x[i] >= MOD) x[i] -= MOD;
        if (y[i] >= MOD) y[i] -= MOD;
      }

      karatsuba(h, x, y, z);
      for (i = 0; i < n; i++) z[i] -= (res[i] + res[i + n]);
      for (i = 0; i < n; i++){
        res[i + h] = (res[i + h] + z[i]) % MOD;
        if (res[i + h] < 0) res[i + h] += MOD;
      }
      ptr -= (h + h + n);
}

/// multiplies two polynomial a(degree n) and b(degree m) and returns the
result modulo MOD in a
/// returns the degree of the multiplied polynomial
/// note that a and b are changed in the process

int mul(int n, int *a, int m, int *b){ /// hash = 903808
   int i, r, c = (n < m ? n : m), d = (n > m ? n : m), *res = buffer +
ptr;
```

```
    r = 1 << (32 - __builtin_clz(d) - (__builtin_popcount(d) == 1));
    for (i = d; i < r; i++) a[i] = b[i] = 0;
    for (i = c; i < d && n < m; i++) a[i] = 0;
    for (i = c; i < d && m < n; i++) b[i] = 0;

    ptr += (r << 1), karatsuba(r, a, b, res), ptr -= (r << 1);
    for (i = 0; i < (r << 1); i++) a[i] = res[i];
    return (n + m - 1);
}

int a[MAX * 2], b[MAX * 2];

int main(){
    int i, j, k, n = MAX - 10;
    for (i = 0; i < n; i++) a[i] = ran(1, 1000000000);
    for (i = 0; i < n; i++) b[i] = ran(1, 991929183);
    clock_t start = clock();
    mul(n, a, n, b);
    dbg(a[n / 2]);
    for (i = 0; i < (n << 1); i++){
        if (a[i] < 0) puts("YO");
    }
    printf("%0.5f\n", (clock() - start) / (1.0 * CLOCKS_PER_SEC));
    return 0;
}
```

Knight_s Tour In Infinite Chessboard.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <assert.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

/// Minimum number of knight moves from (x,y) to (0,0) in non-negative
infinite chessboard
int knight_move(int x, int y){
    int a, b, z, c, d;
    x = abs(x), y = abs(y);
    if (x < y) a = x, x = y, y = a;
    if (x == 2 && y == 2) return 4;
    if (x == 1 && y == 0) return 3;

    if (y == 0 || (y << 1) < x){
        c = y & 1;
        a = x - (c << 1), b = a & 3;
        return ((a - b) >> 1) + b + c;
    }
    else{
        d = x - ((x - y) >> 1);
        z = ((d % 3) != 0), c = (x - y) & 1;
        return ((d / 3) * 2) + c + (z * 2 * (1 - c));
```

```
    }
}

int main(){

}
```

Knuth-Morris-Pratt.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

char str[MAX], pattern[MAX];
int fail[MAX], pos[MAX], T[MAX][26]; /*** T[i]['a'-'z'] = Length of
matched pattern[0:i - 1] + char j ***/

void fail_function(char* str){
    int i, k;
    k = fail[0] = -1;

    for (i = 1; str[i]; i++){
        while (k >= 0 && (str[k + 1] != str[i])) k = fail[k];
        if (str[i] == str[k + 1]) k++;
        fail[i] = k;
    }
}

int count(char* str, char* pattern, int* pos){
    int i, k = 0, len = 0;
    fail_function(pattern);

    for (i = 0; str[i]; i++){
        while (k > 0 && str[i] != pattern[k]) k = fail[k - 1] + 1;
        if (pattern[k] == str[i]) k++;
        if (!pattern[k]){
            pos[len++] = (i - k + 1);
            k = fail[k - 1] + 1;
        }
    }

    return len;
}

void Generate(char* str){
    int i, j, k;
    fail_function(str);

    for (i = 0; str[i]; i++){
        for (j = 0; j < 26; j++){
```

```
            char ch = j + 'a';

            k = i;
            while (k > 0 && str[k] != ch) k = fail[k - 1] + 1;
            if (str[k] == ch) k++;
            T[i][j] = k;
        }
    }
}

int main(){
}
```

LCA Extended.cpp

```
#include <bits/stdtr1c++.h>

#define LOG 20
#define MAX 200010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// LCA of undirected weighted tree, 0-based index
namespace lca{ /// hash = 203060
    long long sum[MAX];
    vector <int> ar[MAX], weight[MAX];
    int n, r, parent[MAX], depth[MAX], lg[MAX], dp[MAX][LOG];

    void init(int nodes, int root){
        n = nodes, r = root, lg[0] = lg[1] = 0;
        for (int i = 2; i <= n; i++) lg[i] = lg[i >> 1] + 1;
        for (int i = 0; i <= n; i++) ar[i].clear(), weight[i].clear();
    }

    void add_edge(int u, int v, int w){
        ar[u].push_back(v), weight[u].push_back(w);
        ar[v].push_back(u), weight[v].push_back(w);
    }

    int lca(int a, int b){
        if (a == b) return a;
        if (depth[a] < depth[b]) swap(a, b);

        for (int i = lg[depth[a] - depth[b]]; i >= 0; i--){
            if ((depth[a] - (1 << i)) >= depth[b]) a = dp[a][i];
        }
        if (a == b) return a;

        for (int i = lg[depth[a]]; i >= 0; i--){
            if (dp[a][i] != dp[b][i]){
```

```
                a = dp[a][i];
                b = dp[b][i];
            }
        }

        return (a == b) ? a : parent[a];
    }

    long long dis(int u, int v){
        int l = lca(u, v);
        long long res = sum[u] + sum[v] - (sum[l] << 1LL);
        return res;
    }

    void dfs(int i, int p){
        int j, len = ar[i].size();
        for (j = 0, parent[i] = p; j < len; j++){
            if (ar[i][j] != p){
                sum[ar[i][j]] = sum[i] + weight[i][j];
                depth[ar[i][j]] = depth[i] + 1;
                dfs(ar[i][j], i);
            }
        }
    }

    void build(){
        depth[r] = 0, sum[r] = 0;
        dfs(r, r);

        for (int l = 0; l <= lg[n]; l++){
            for (int i = 0; i < n; i++){
                if (!l) dp[i][l] = parent[i];
                else dp[i][l] = dp[dp[i][l - 1]][l - 1];
            }
        }
    }
}

int main(){
    return 0;
}
```

LCA OP.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// LCA in O(n)
```

```
/// 0-based index and for undirected trees. To change for directed trees
just modify add() function
/// Call init(nodes, root (set to any for undirected) ), add edges, call
build() to pre-process
/// Note that MAX in Sparse Table must be two times MAX nodes
/// Don't forget to call build()! If Program crashes make sure build() is
called before any queries

namespace lca{
   #define MAX     100010 /// Maximum number of nodes in the tree
   #define LOG         18 /// (int)floor(log2(MAX)) + 2,  {...7 = 4, 8 =
5, 9 = 5...}
   #define MAXB     11188 /// ((MAX * 2) + LG - 1) / LG, ((MAX * 2) / LG)
+ 77 works fine
   #define op(a, b) ((lca::ar[(a)]) < (lca::ar[(b)]) ? (a) : (b)) ///
comparator of two numbers, set to min

   typedef pair<int, int> Pair; /// Next node and Edge weight

   int n, r;
   vector <Pair> adj[MAX]; /// Consider removing weight if only LCA is
required
   int m, lg, len, T[MAX << 1], ar[MAX << 1], tour[MAX << 1], first[MAX
<< 1], mask[1 << LOG], dp[LOG][MAXB];

   void init(int nodes, int root){
       n = nodes, r = root;
       for (int i = 0; i < MAX; i++) adj[i].clear();
   }

   /// Adds undirected edge from a-b with edge weight w (0 based index)
   void add(int a, int b, int w){
       adj[a].push_back(Pair(b, w));
       adj[b].push_back(Pair(a, w));
   }

   void dfs(int i, int p, int h){
       int j, x, len = adj[i].size();
       ar[m] = h, tour[m] = i, first[i] = m++;

       for (j = 0; j < len; j++){
           x = adj[i][j].first;
           if (x != p){
               dfs(x, i, h + 1);
               ar[m] = h, tour[m++] = i;
           }
       }
   }

   void build(){
       m = 0;
       dfs(r, -1, 0);
       int i, j, k, d, top;
```

```
        lg = 32 - __builtin_clz(m);
        d = -1, i = 0, len = (m + lg - 1) / lg;

        while (i < m){
            dp[0][++d] = i++;
            for (j = 1; j < lg && i < m; i++, j++){
                dp[0][d] = op(i, dp[0][d]);
            }
        }

        for (j = 1; j < lg; j++){
            d = (1 << j) >> 1;
            for (i = 0; i < len; i++){
                dp[j][i] = op(dp[j - 1][i], dp[j - 1][i + ((i + d) < len ?
d : 0)]);
            }
        }

        for (i = 0; i < len; i++){
            top = 0, d = (i * lg) + lg;
            for (j = d - lg; j < m && j < d; j++){
                while (top && ar[j] < ar[mask[top]]) top--;
                T[j] = 1 << (d - j - 1);
                if (top) T[j] |= T[mask[top]];
                mask[++top] = j;
            }
        }

        for (i = 1, k = 1 << lg, d = lg - 1; i < k; i++){
            if (i >= (1 << (lg - d))) d--;
            mask[i] = d;
        }
    }

    /// returns the lowest common ancestors of l and r
    inline int lca(int l, int r){
        l = first[l], r = first[r];
        if (l > r) swap(l, r);
        int c, d, x = (l / lg) + 1, y = (r / lg) - 1, res = l;

        if(x <= y){
            d = lg - mask[y - x + 1] - 1;
            res = op(res, op(dp[d][x], dp[d][y - (1 << d) + 1]));
        }

        c = x * lg, d = y * lg;
        res = op(res, mask[T[(c - 1) < r ? (c - 1) : r] & (~(((1 << (l - c
+ lg )) - 1) << (c - l)))] + c - lg);
        l = l > (d + lg) ? l : d + lg;
        res = op(res, mask[T[r] & (~(((1 << (l - d - lg)) - 1) << (d + (lg
<< 1) - l)))] + d + lg);
        return tour[res];
    }
}
```

```
int main(){
   int n, q, i, j, k, a, b;

   while (scanf("%d", &n) != EOF){
       lca::init(n, 0);
       for (i = 0; i < n; i++){
           scanf("%d", &k);
           while (k--){
               scanf("%d", &b);
               lca::add(i, b, 1);
           }
       }
       lca::build();

       scanf("%d", &q);
       while (q--){
           scanf("%d %d", &a, &b);
           printf("%d\n", lca::lca(a, b));
       }
   }
   return 0;
}
```

LCA.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define LOG 20
#define MAX 200010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

namespace lca{ /// LCA of undirected tree, 0-based index
   vector <int> ar[MAX];
   int n, r, parent[MAX], depth[MAX], lg[MAX], dp[MAX][LOG];

   void init(int nodes, int root){
       n = nodes, r = root, lg[0] = lg[1] = 0;
       for (int i = 0; i <= n; i++) ar[i].clear();
       for (int i = 2; i <= n; i++) lg[i] = lg[i >> 1] + 1;
   }

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

   int lca(int a, int b){
       if (a == b) return a;
```

```
        if (depth[a] < depth[b]) swap(a, b);

        for (int i = lg[depth[a] - depth[b]]; i >= 0; i--){
            if ((depth[a] - (1 << i)) >= depth[b]) a = dp[a][i];
        }
        if (a == b) return a;

        for (int i = lg[depth[a]]; i >= 0; i--){
            if (dp[a][i] != dp[b][i]){
                a = dp[a][i];
                b = dp[b][i];
            }
        }

        return (a == b) ? a : parent[a];
    }

    void dfs(int i, int p){
        int j, len = ar[i].size();
        for (j = 0, parent[i] = p; j < len; j++){
            if (ar[i][j] != p){
                depth[ar[i][j]] = depth[i] + 1;
                dfs(ar[i][j], i);
            }
        }
    }

    void build(){
        depth[r] = 0;
        dfs(r, r);

        for (int l = 0; l <= lg[n]; l++){
            for (int i = 0; i < n; i++){
                if (!l) dp[i][l] = parent[i];
                else dp[i][l] = dp[dp[i][l - 1]][l - 1];
            }
        }
    }
}

int main(){

}

LCIS.c
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 2057
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
```

```
/// Longest common increasing subsequence of A and B in O(n * m)
int lcis(int n, int* A, int m, int* B){
   int i, j, l, res, dp[MAX] = {0};

   for (i = 0; i < n; i++){
       for (l = 0, j = 0; j < m; j++){
           if (A[i] == B[j] && dp[j] <= l) dp[j] = l + 1;
           else if (B[j] < A[i] && dp[j] > l) l = dp[j];
       }
   }

   for (i = 0, res = 0; i < m; i++){
       if (dp[i] > res) res = dp[i];
   }
   return res;
}

int main(){

}
```

LIS + LDS.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

int* LIS(int n, int ar[]){
   int* lis = new int[n + 10];
   int* val = new int[n + 10];

   int i, d, len = 0;
   lis[0] = 1, val[len++] = ar[0];

   for (i = 1; i < n; i++){
       d = lower_bound(val, val + len, ar[i]) - &val[0];
       lis[i] = d + 1;
       if (d == len) val[len++] = ar[i];
       else val[d] = ar[i];
   }

   return lis;
}

int* LDS(int n, int ar[]){
   for (int i = 0; i < n; i++) ar[i] = INT_MAX - ar[i];
   reverse(ar, ar + n);
   int* lds = LIS(n, ar);
```

```
    reverse(lds, lds + n);
    reverse(ar, ar + n);
    for (int i = 0; i < n; i++) ar[i] = INT_MAX - ar[i];

    return lds;
}

int main(){
    int n, i, j, k, m;

    while (scanf("%d", &n) != EOF){
        int* ar = new int[n + 10];
        for (int i = 0; i < n; i++) scanf("%d", &ar[i]);

        int* lis = LIS(n, ar);
        int* lds = LDS(n, ar);
    }
    return 0;
}
```

Lagrange_s Polynomial Interpolation.py

```
import sys
import math

val = [0, 1, 2, 3, 4, 5, 6]
out = [1, 1, 2, 4, 8, 16, 31]

def lagrange(n):
      res = 0
      for i in range(len(val)):
            x = out[i]
            y = 1
            for j in range(len(val)):
                  if (i != j):
                        x *= (n - val[j])
                        y *= (val[i] - val[j])

            res += (x // y)
      return res

def main():
      print (lagrange(20))

if __name__ == '__main__':
   main()
```

Lagrnage_s Polynomial Interpolation.py

```
import sys
import math

val = [0, 1, 2, 3, 4, 5, 6]
```

```
out = [1, 1, 2, 4, 8, 16, 31]

def lagrange(n):
      res = 0
      for i in range(len(val)):
            x = out[i]
            y = 1
            for j in range(len(val)):
                  if (i != j):
                        x *= (n - val[j])
                        y *= (val[i] - val[j])

            res += (x // y)
      return res

def main():
      print (lagrange(20))

if __name__ == '__main__':
   main()
```

Lucas Theorem Extended.cpp

```
#include <bits/stdtr1c++.h>

#define MAXP 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/// Lucas theorem to calculate binomial co-efficients modulo a prime

namespace lc{
   int MOD = 1000000007;
   int fact[MAXP], inv[MAXP];

   /// Call once with the modulo prime
   void init(int prime){
       MOD = prime;
       fact[0] = 1, inv[MOD - 1] = MOD - 1;
       for (int i = 1; i < MOD; i++) fact[i] = ((long long)fact[i - 1] *
i) % MOD;
       for (int i = MOD - 2; i >= 0; i--) inv[i] = ((long long)inv[i + 1]
* (i + 1)) % MOD;
   }

   inline int count(int n, int k){
       if (k > n) return 0;
       int x = ((long long)inv[n - k] * inv[k]) % MOD;
       return ((long long)x * fact[n]) % MOD;
   }
```

```
    /// Lucas theorem, calculates binomial(n, k) modulo MOD, MOD must be a
prime
    inline int binomial(long long n, long long k){
        if (k > n) return 0;

        int res = 1;
        k = min(k, n - k);
        while (k && res){
            res = ((long long)res * count(n % MOD, k % MOD)) % MOD;
            n /= MOD, k /= MOD;
        }
        return res;
    }

    /*** Alternate and extended functionalities ***/

    /// Must call init with prime before (Or set lc::MOD = prime)
    /// Computes (n! / (p ^ (n / p))) % p in O(p log(n)) time, p MUST be a
prime
    /// That is, calculating n! without p's powers
    /// For instance factmod(9, 3) = (1 * 2 * 4 * 5 * 2 * 7 * 8 * 1) % 3 =
1
    inline int factmod(long long n, int p){
        int i, res = 1;
        while (n > 1) {
            if ((n / p) & 1) res = ((long long)res * (p - 1)) % p;
            for (i = n % p; i > 1; i--) res = ((long long)res * i) % p;
            n /= p;
        }
        return (res % p);
    }

    inline int expo(int a, int b){
        int res = 1;

        while (b){
            if (b & 1) res = (long long)res * a % MOD;
            a = (long long)a * a % MOD;
            b >>= 1;
        }
        return res;
    }

    /// Trailing zeros of n! in base p, p is a prime
    inline long long fact_ctz(long long n, long long p){
        long long x = p, res = 0;
        while (n >= x){
            res += (n / x);
            x *= p;
        }
        return res;
    }

    /// Calculates binomial(n, k) modulo MOD, MOD must be a prime
```

```
    inline int binomial2(long long n, long long k){
        if (k > n) return 0;
        if (fact_ctz(n, MOD) != (fact_ctz(k, MOD) + fact_ctz(n - k, MOD)))
return 0;
        int a = factmod(n - k, MOD), b = factmod(k, MOD), c = factmod(n,
MOD);
        int x = ((long long)expo(a, MOD - 2) * expo(b, MOD - 2)) % MOD;
        return ((long long)x * c) % MOD;
    }
}

int main(){
    lc::init(997);
    printf("%d\n", lc::binomial(10, 5));
    printf("%d\n", lc::binomial(1996, 998));

    lc::MOD = 10007;
    printf("%d\n", lc::binomial2(10, 5));
    printf("%d\n", lc::binomial2(1996, 998));
    return 0;
}
```

Lucas Theorem.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAXP 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/// Lucas theorem to calculate binomial co-efficients modulo a prime

namespace lc{
    int MOD = 1000000007;
    int fact[MAXP], inv[MAXP];

    /// Call once with the modulo prime
    void init(int prime){
        MOD = prime;
        fact[0] = 1, inv[MOD - 1] = MOD - 1;
        for (int i = 1; i < MOD; i++) fact[i] = ((long long)fact[i - 1] *
i) % MOD;
        for (int i = MOD - 2; i >= 0; i--) inv[i] = ((long long)inv[i + 1]
* (i + 1)) % MOD;
    }

    inline int count(int n, int k){
        if (k > n) return 0;
        int x = ((long long)inv[n - k] * inv[k]) % MOD;
        return ((long long)x * fact[n]) % MOD;
    }
```

```
   /// Lucas theorem, calculates binomial(n, k) modulo MOD, MOD must be a
prime
   inline int binomial(long long n, long long k){
       if (k > n) return 0;

       int res = 1;
       while (k){
           res = ((long long)res * count(n % MOD, k % MOD)) % MOD;
           n /= MOD, k /= MOD;
       }
       return res;
   }
}

int main(){
   lc::init(997);
   printf("%d\n", lc::binomial(10, 5));
   printf("%d\n", lc::binomial(1996, 998));
   return 0;
}
```

Lucas.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int t, a, b;
long long int p, factorial[MAX], inv[MAX], A[100], B[100], temp[100];

long long int P(int x, int n, int MOD){
   if (n == 0) return 1;
   else if (n & 1) return ((P(x, n - 1, MOD) * x) % MOD);
   else{
       long long int y = P(x, n >> 1, MOD);
       return ((y * y) % MOD);
   }
}

long long int Lucas(long long int n, long long int k){
   int i, j;
   long long int x, y, z, m, r;

   factorial[0] = 1;
   for (i = 0; i < p; i++){
       if (i) factorial[i] = (factorial[i - 1] * i) % p;
   }

   a = 0, i = 0, x = n;
```

```
    for (; ;){
        temp[i++] = (x % p);
        x /= p;
        if (x == 0) break;
    }
    for (j = i - 1; j >= 0; j--) A[a++] = temp[j];

    b = 0, i = 0, x = k;
    for (; ;){
        temp[i++] = (x % p);
        x /= p;
        if (x == 0) break;
    }
    for (j = i - 1; j >= 0; j--) B[b++] = temp[j];

    long long int res = 1;
    for (j = b - 1, i = a - 1; i >= 0; i--, j--){
        m = A[i];
        if (j < 0) r = 0;
        else r = B[j];

        if ((m - r) < 0){
            res = 0;
            break;
        }

        x = factorial[m];
        z = (factorial[r] * factorial[m - r]) % p;
        y = P(z, p - 2, p);
        long long int v = (x * y) % p;
        res = (res * v) % p;
    }

    return res;
}

int main(){
    int i, j;
    long long int n, k;

    while (scanf("%d", &t) != EOF){
        while (t--){
            scanf("%lld %lld %lld", &n, &k, &p);
            long long int res = Lucas(n + 1, n - k);
            printf("%lld\n", res);
        }
    }
    return 0;
}
```

MST.cpp

```
#include <bits/stdtr1c++.h>
```

```
#define MAXN 200010
#define MAXE 200010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

struct Edge{
   int u, v, cost;

   Edge(){
   }

   Edge(int a, int b, int c){
       u = a, v = b, cost = c;
   }

   bool operator < (const Edge& other) const{
       return (cost < other.cost);
   }
};

int n, m;
struct Edge E[MAXE];
int union_parent[MAXN], union_rank[MAXN];

int find_root(int i){
   while (i != union_parent[i]){
       union_parent[i] = union_parent[union_parent[i]];
       i = union_parent[i];
   }
   return union_parent[i];
}

int main(){
   int i, j, k, a, b, c, d, sum, counter, mst;

   while (scanf("%d %d", &n, &m) != EOF){
       if (n == 0 && m == 0) break;

       for (i = 0; i <= n; i++){
           union_rank[i] = 0;
           union_parent[i] = i;
       }

       sum = 0;
       for (i = 0; i < m; i++){
           scanf("%d %d %d", &E[i].u, &E[i].v, &E[i].cost);
           sum += E[i].cost;
       }
       sort(E, E + m);

       counter = 0, mst = 0;
```

```
        for (i = 0; i < m; i++){
            a = E[i].u, b = E[i].v;

            c = find_root(a), d = find_root(b);
            if (c != d){
                if (union_rank[c] < union_rank[d]) union_parent[c] = d;
                else if (union_rank[c] > union_rank[d]) union_parent[d] =
c;
                else{
                    union_parent[c] = d;
                    union_rank[d]++;
                }

                counter++;
                mst += E[i].cost;
                if (counter == (n - 1)) break;
            }
        }

        int res = sum - mst;
        printf("%d\n", res);
    }
    return 0;
}
```

Macros.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

#define gen() freopen("lol.txt", "w", stdout)
#define write() freopen("output.txt", "w", stdout)
#define test1() freopen("test1.txt", "w", stdout)
#define test2() freopen("test2.txt", "w", stdout)
#define rand(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

#define min(a,b) ((a)<(b) ? (a):(b))
#define max(a,b) ((a)>(b) ? (a):(b))
#define swap(a,b) ((a) = (a) + (b) - (b = a))

#define setbit(x, i) ((x) | (1 << (i)))
#define pow2(x) ((x) && !((x) & ((x) - 1)))
#define resetbit(x, i) ((x) & (~(1 << (i))))
#define getbit(x, i) (((x) & (1 << (i))) ? (1):(0))

const int dx[] = {0, 0, -1, 1};
const int dy[] = {-1, 1, 0, 0};
const int dx[] = {0, 0, -1, 1, -1, -1, 1, 1};
const int dy[] = {-1, 1, 0, 0, -1, 1, -1, 1};
const int knightx[] = {-1, -1, 1, 1, -2, -2, 2, 2};
```

```
const int knighty[] = {-2, 2, -2, 2, -1, 1, -1, 1};
#define valid(i, j) ((i) >= 0 && (i) < n && (j) >= 0 && (j) < n)
#define valid(i, j) ((i) >= 0 && (i) < n && (j) >= 0 && (j) < m)

/************************************
swap works perfectly for any signed integer, most probably works great
for doubles too
pow2 works perfectly for any signed integer, except for -(2^31) = -
2147483648
************************************/

int main(){

}
```

Macros.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

#define gen() freopen("lol.txt", "w", stdout)
#define write() freopen("output.txt", "w", stdout)
#define test1() freopen("test1.txt", "w", stdout)
#define test2() freopen("test2.txt", "w", stdout)

#define setbit(x, i) ((x) | (1 << (i)))
#define pow2(x) ((x) && !((x) & ((x) - 1)))
#define resetbit(x, i) ((x) & (~(1 << (i))))
#define getbit(x, i) (((x) & (1 << (i))) ? (1):(0))

#define printbits(x, n) cout << #x << " = " << x << " = " << bitset<n>(x)
<< endl /// Least significant n bits of x, n must be constant
#define tobinary(x) string(bitset<64>(x).to_string<char,
string::traits_type, string::allocator_type>()).substr(min(63,
__builtin_clzll(x)), 64)
#define lastbits(x, n) cout << string(bitset<64>(x).to_string<char,
string::traits_type, string::allocator_type>()).substr(64 - n, 64) <<
endl
#define firstbits(x, n) cout << string(bitset<64>(x).to_string<char,
string::traits_type, string::allocator_type>()).substr(min(63,
__builtin_clzll(x)), 64).substr(0, n) << endl;

using namespace std;

int main(){
   cout <<  fixed << setprecision(25) << res << endl;
}
```

Manacher_s Algorithm.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 1010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int n, m, L[(MAX << 1) + 5];
char str[MAX], T[(MAX << 1) + 5];

void Manacher(char* str){
   int i, c, r, l;
   n = strlen(str), m = (n << 1) + 3;
   T[0] = 64, T[m - 1] = 36, T[m - 2] = 35, T[m] = 0;

   for (i = 0; i < n; i++){
       T[(i << 1) + 1] = 35;
       T[(i + 1) << 1] = str[i];
   }

   c = r = L[0] = 0;
   for (i = 1; i < m; i++){
       L[i] = 0;
       l = (c << 1) - i;
       if (r > i) L[i] = (L[l]<(r - i)) ? L[l]:(r - i);
       while (T[i + 1 + L[i]] == T[i - 1 - L[i]]) L[i]++;

       if ((i + L[i]) > r){
           c = i;
           r = i + L[i];
       }
   }
}

int main(){
   while (scanf("%s", str) != EOF){
       Manacher(str);
   }
   return 0;
}
```

Manacher_s Algorithm.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/*** Manacher's algorithm to generate longest palindromic substrings for
all centers ***/
```

```
/// When i is even, pal[i] = largest palindromic substring centered from
str[i / 2]
/// When i is odd, pal[i] = largest palindromic substring centered
between str[i / 2] and str[i / 2] + 1

vector <int> manacher(char *str){ /// hash = 784265
   int i, j, k, l = strlen(str), n = l << 1;
   vector <int> pal(n);

   for (i = 0, j = 0, k = 0; i < n; j = max(0, j - k), i += k){
       while (j <= i && (i + j + 1) < n && str[(i - j) >> 1] == str[(i +
j + 1) >> 1]) j++;
       for (k = 1, pal[i] = j; k <= i && k <= pal[i] && (pal[i] - k) !=
pal[i - k]; k++){
           pal[i + k] = min(pal[i - k], pal[i] - k);
       }
   }

   pal.pop_back();
   return pal;
}

int main(){
   char str[100];
   while (scanf("%s", str)){
       auto v = manacher(str);
       for (auto it: v) printf("%d ", it);
       puts("");
   }
   return 0;
}
```

Matrix Expo.cpp

```
--------------------------------------------------
/***

Sometimes we may need to maintain more than one recurrence, where they
are interrelated.
For example, let a recurrence relation be:
g(n) = 2g(n-1) + 2g(n-2) + f(n), where, f(n) = 2f(n-1) + 2f(n-2).
Here, recurrence g(n) is dependent upon f(n) and the can be calculated in
the same matrix
but of increased dimensions. Lets design the matrices A, B then we'll try
to find matrix M.


                       Matrix A               Matrix B

                       |  g(n)   |             | g(n+1)  |
                       | g(n-1)  |             |  g(n)   |
                       | f(n+1)  |             | f(n+2)  |
                       |  f(n)   |             | f(n+1)  |
```

Here, g(n+1) = 2g(n) + 2g(n-1) + f(n+1) and f(n+2) = 2f(n+1) + 2f(n).
Now, using the above process, we can generate the objective matrix M as follows:

```
| 2  2  1  0 |
| 1  0  0  0 |
| 0  0  2  2 |
| 0  0  1  0 |

***/


#include <bits/stdtr1c++.h>

#define MAXN 10
#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

struct Matrix{
   int n;
   long long ar[MAXN][MAXN];

   Matrix(){
   }

   Matrix(int x){
       n = x;
       clr(ar);
   }
};

struct Matrix mul(struct Matrix& A, struct Matrix& B){
   int n = A.n;
   struct Matrix C = Matrix(n);

   int i, j, k;
   long long res = 0;
   for (i = 0; i < n; i++){
       for (j = 0; j < n; j++){
           for (k = 0, res = 0; k < n; k++){
               res = res + (A.ar[i][k] * B.ar[k][j]);
               if (res >= MOD) res %= MOD;
           }
           C.ar[i][j] = res;
       }
   }

   return C;
}
```

```
struct Matrix pow(struct Matrix mat, long long n){
   int c = 0;
   struct Matrix res = Matrix(mat.n);
   for (int i = 0; i < mat.n; i++) res.ar[i][i] = 1;

   while (n){
       if (n & 1LL){
           if (!c++) res = mat;
           else res = mul(res, mat);
       }

       n >>= 1LL;
       mat = mul(mat, mat);
   }

   return res;
}

long long matrix_expo(int n, struct Matrix mat, long long val[MAXN], long
long p){
   if (p < n) return (val[p] % MOD);

   int i, j;
   long long res = 0;
   Matrix pw = pow(mat, p - n + 1);

   for (i = 0; i < n; i++){
       long long x = (pw.ar[0][i] * val[n - i - 1]) % MOD;
       res = res + x;
   }
   return (res % MOD);
}

int main(){
   int n = 2;
   struct Matrix mat = Matrix(n);

   mat.ar[0][0] = mat.ar[0][1] = 1, mat.ar[1][0] = 1, mat.ar[1][1] = 0;
   long long val[] = {0, 1};

   long long res = matrix_expo(n, mat, val, 13);
   dbg(res);
   return 0;
}
```

Matrix.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
```

```
using namespace std;

struct Matrix{
   int row, col;
   int ar[101][101]; /// Change matrix size here, also change to long
long for safety

   Matrix(){} ///Beware if matrix can contain negative numbers in matrix
exponentiation problems
   Matrix(int n, int m, int diagonal = 0){
        clr(ar);
        row = n, col = m;
        for (int i = min(n, m) - 1; i >= 0; i--) ar[i][i] = diagonal;
   }

   /// To multiply two matrices A and B, the number of columns in A must
equal the number of rows in B
   Matrix operator* (const Matrix& other) const{ /// hash = 709758
        int i, j, k;
            Matrix res(row, other.col);
            long long x, y = (long long)MOD * MOD;

            for(i = 0; i < row; i++){
                  for(j = 0; j < other.col; j++){
                        for(k = 0, x = 0; k < col; k++){
                              x += ((long long)ar[i][k] * other.ar[k][j]);
/// replace matrix other with its transpose matrix to reduce cache miss
                              if (x >= y) x -= y;
                        }
                        res.ar[i][j] = x % MOD;
                  }
            }
            return res;
      }

      Matrix operator^ (long long n) const{
          Matrix x = *this, res = Matrix(row, col, 1);
            while (n){
                  if (n & 1) res = res * x;
                  n = n >> 1, x = x * x;
            }
            return res;
      }

   /// Transpose matrix, T[i][j] = ar[j][i]
      Matrix transpose(){
          Matrix res = Matrix(col, row);
        for (int i = 0; i < row; i++){
            for (int j = 0; j < col; j++){
                res.ar[j][i] = ar[i][j];
            }
        }
        return res;
      }
```

```
      /// rotates the matrix 90 degrees clockwise
      Matrix rotate(){
       Matrix res = this->transpose();
       for (int i = 0; i < res.row; i++) reverse(res.ar[i], res.ar[i] +
res.col);
       return res;
      }

      inline void print(){
          for (int i = 0; i < row; i++){
           for (int j = 0; j < col; j++){
               printf("%d%c", ar[i][j], ((j + 1) == col) ? 10 : 32);
           }
          }
      }
};

int main(){
   Matrix a = Matrix(4, 5, 1);
   int k = 0;
   for (int i = 0; i < a.row; i++){
       for (int j = 0; j < a.col; j++){
           a.ar[i][j] = ++k;
       }
   }
   a.print();
   puts("");
   Matrix b = a.rotate();
   b.print();
   return 0;

   Matrix x = Matrix(5, 5, 5);
   Matrix y = x ^ 5;
   x.print();
   y.print();
}


/***

Sometimes we may need to maintain more than one recurrence, where they
are interrelated.
For example, let a recurrence relation be:
g(n) = 2g(n-1) + 2g(n-2) + f(n), where, f(n) = 2f(n-1) + 2f(n-2).
Here, recurrence g(n) is dependent upon f(n) and the can be calculated in
the same matrix
but of increased dimensions. Lets design the matrices A, B then we'll try
to find matrix M.


                       Matrix A               Matrix B

                       |  g(n)  |              | g(n+1) |
```

```
                    | g(n-1) |              |  g(n)  |
                    | f(n+1) |              | f(n+2) |
                    |  f(n)  |              | f(n+1) |


Here, g(n+1) = 2g(n) + 2g(n-1) + f(n+1) and f(n+2) = 2f(n+1) + 2f(n).
Now, using the above process, we can generate the objective matrix M as
follows:

| 2  2  1  0 |
| 1  0  0  0 |
| 0  0  2  2 |
| 0  0  1  0 |


/// Matrix Rotations:

Rotate by +90:
Transpose
Reverse each row

Rotate by -90:
Transpose
Reverse each column


Rotate by +180:
Method 1: Rotate by +90 twice
Method 2: Reverse each row and then reverse each column

Rotate by -180:
Method 1: Rotate by -90 twice
Method 2: Reverse each column and then reverse each row
Method 3: Reverse by +180 as they are same

***/
```

Maximum Divisors.c

```
#include <stdio.h>

unsigned long long n, res, idx;
int p, primes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
47, 53, 59, 61, 67, 71};

unsigned long long mul(unsigned long long a, unsigned long long b){
   unsigned long long res = 0;

   while (b){
       if (b & 1LL) res = (res + a);
       if (res > n) return 0;
       a = (a << 1LL);
       b >>= 1LL;
   }
```

```
    return res;
}

void backtrack(int i, int lim, unsigned long long val, unsigned long long
r){
    if ((r > res) || (r == res && val < idx)) res = r, idx = val;
    if (i == p) return;

    int d;
    unsigned long long x = val;

    for (d = 1; d <= lim; d++){
        x = mul(x, primes[i]);
        if (x == 0) return;
        backtrack(i + 1, d, x, r * (d + 1));
    }
}

int main(){
    /* Tested for n <= 10^18 */

    p = sizeof(primes) / sizeof(int);

    while (scanf("%llu", &n) != EOF){
        res = 0;
        backtrack(0, 100, 1, 1);
        printf("%llu = %llu\n", idx, res);
    }
    return 0;
}
```

Maximum Matching in General Graphs (Randomized Algorithm).c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>

#define MAX 1010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

bool adj[MAX][MAX];
int n, ar[MAX][MAX];
const int MOD = 1073750017;

int expo(long long x, int n){
    long long res = 1;

    while (n){
        if (n & 1) res = (res * x) % MOD;
        x = (x * x) % MOD;
        n >>= 1;
```

```
    }

    return (res % MOD);
}

int rank(int n){ /// hash = 646599
    long long inv;
    int i, j, k, u, v, x, r = 0, T[MAX];

    for (j = 0; j < n; j++){
        for (k = r; k < n && !ar[k][j]; k++){}
        if (k == n) continue;

        inv = expo(ar[k][j], MOD - 2);
        for (i = 0; i < n; i++){
            x = ar[k][i];
            ar[k][i] = ar[r][i];
            ar[r][i] = (inv * x) % MOD;
        }

        for (u = r + 1; u < n; u++){
            if (ar[u][j]){
                for (v = j + 1; v < n; v++){
                    if (ar[r][v]){
                        ar[u][v] = ar[u][v] - (((long long)ar[r][v] *
ar[u][j]) % MOD);
                        if (ar[u][v] < 0) ar[u][v] += MOD;
                    }
                }
            }
        }
        r++;
    }

    return r;
}

int tutte(int n){
    int i, j;
    srand(time(0));

    clr(ar);
    for (i = 0; i < n; i++){
        for (j = i + 1; j < n; j++){
            if (adj[i][j]){
                unsigned int x = (rand() << 15) ^ rand();
                x = (x % (MOD - 1)) + 1;
                ar[i][j] = x, ar[j][i] = MOD - x;
            }
        }
    }

    return (rank(n) >> 1);
}
```

```
int main(){
   int T = 0, t, m, i, j, a, b;

   scanf("%d", &t);
   while (t--){
       clr(adj);
       scanf("%d %d", &n, &m);
       while (m--){
           scanf("%d %d", &a, &b);
           a--, b--;
           adj[a][b] = adj[b][a] = true;
       }

       printf("Case %d: %d\n", ++T, tutte(n));
   }
   return 0;
}
```

Maximum Square + Diamond.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 505
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

char str[MAX];
bool ar[MAX][MAX];
int n, m, dp[MAX][MAX] = {0};

int Square(){ /// hash = 270386
   int i, j, x, y, res, counter = 0;

   for (i = 0; i < MAX; i++){
       dp[0][i] = dp[i][0] = dp[n + 1][i] = dp[i][m + 1] = 0;
   }

   for (i = 1; i <= n; i++){
       for (j = 1; j <= m; j++){
           if (ar[i][j]){
               x = dp[i - 1][j], y = dp[i][j - 1];
               if (y < x) x = y;
               res = x + ar[i - x][j - x];
               dp[i][j] = res;

               res--;
               if (res) counter += res;
           }
           else dp[i][j] = 0;
       }
   }
```

```
    return counter;
}

int Kite(){ /// hash = 775619
    int i, j, x, y, res, counter = 0;

    for (i = 0; i < MAX; i++){
        dp[0][i] = dp[i][0] = dp[n + 1][i] = dp[i][m + 1] = 0;
    }

    for (i = 1; i <= n; i++){
        for (j = 1; j <= m; j++){
            if (ar[i][j]){
                x = dp[i - 1][j - 1], y = dp[i - 1][j + 1];
                if (y < x) x = y;
                y = x << 1;

                if (!x || !ar[i - 1][j]) res = 1;
                else if (ar[i - y][j] && ar[i - y + 1][j]) res = x + 1;
                else res = x;
                dp[i][j] = res;

                res--;
                if (res) counter += res;
            }
            else dp[i][j] = 0;
        }
    }
    return counter;
}

int main(){
    int i, j, k;

    while (scanf("%d", &n) != EOF){

        m = n;
        for (i = 1; i <= n; i++){
            scanf("%s", str);
            for (j = 0; str[j] != 0; j++){
                ar[i][j + 1] = (str[j] == 'x');
            }
        }

        int x = Square();
        int y = Kite();
        int res = x + y;
        printf("%d\n", res);
    }
    return 0;
}
```

Maximum XOR Subest.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define bitlen(x) ((x) == 0 ? (0) : (64 - __builtin_clzll(x)))
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

long long ar[MAX];

long long solve(int n, long long* ar){ /// hash = 220650
   vector <long long> v[64];
   for (int i = 0; i < n; i++) v[bitlen(ar[i])].push_back(ar[i]);

   long long m, x, res = 0;
   for (int i = 63; i > 0; i--){
       int l = v[i].size();
       if (l){
           m = v[i][0];
           res = max(res, res ^ m);

           for (int j = 1; j < l; j++){
               x = m ^ v[i][j];
               if (x) v[bitlen(x)].push_back(x);
           }
           v[i].clear();
       }
   }
   return res;
}

int main(){
   return 0;
}
```

Merge Sort Tree.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

int n, ar[MAX];
vector<int> tree[MAX << 2];

void merge_sort_tree(int idx, int a, int b, int* ar){ /// hash = 974987
```

```
   int p = idx << 1, q = p | 1, c = (a + b) >> 1, d = c + 1;
   int i = 0, j = 0, k = 0, u = c - a + 1, v = b - d + 1, len = b - a +
1;

   tree[idx].resize(len, 0);
   if (a == b){
       tree[idx][0] = ar[a];
       return;
   }

   merge_sort_tree(p, a, c, ar);
   merge_sort_tree(q, d, b, ar);
   while (len--){
       if (i == u) tree[idx][k++] = tree[q][j++];
       else if (j == v) tree[idx][k++] = tree[p][i++];
       else if (tree[p][i] < tree[q][j]) tree[idx][k++] = tree[p][i++];
       else tree[idx][k++] = tree[q][j++];
   }
}

/// Count of numbers <= k in the segment l-r
inline int query(int idx, int a, int b, int l, int r, int k){ /// hash =
476541
   int p = idx << 1, q = p | 1;
   int c = (a + b) >> 1, d = c + 1;

   if (a == l && b == r){
       if (tree[idx][0] > k) return 0;
       else return upper_bound(tree[idx].begin(), tree[idx].end(), k) -
tree[idx].begin();
   }
   if (r <= c) return query(p, a, c, l, r, k);
   else if (l >= d) return query(q, d, b, l, r, k);
   else return query(p, a, c, l, c, k) + query(q, d, b, d, r, k);
}

int main(){

}
```

Miller Rabin (Deterministic + Integer).cpp

```
#include <bits/stdtr1c++.h>

#define MAX 1000010
#define isprime(x) prm::miller_rabin(x)
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

namespace prm{ /// hash = 663918
   bitset <MAX> flag;
```

```
int p = 0, prime[78777];
const unsigned long long base[] = {4230279247111683200ULL,
14694767155120705706ULL, 16641139526367750375ULL};

void Sieve(){
    int i, j, x;
    for (i = 3; i < MAX; i += 2) flag[i] = true;
    for (i = 3, flag[2] = true; (i * i) < MAX; i += 2){
        if (flag[i]){
            for (j = (i * i), x = i << 1; j < MAX; j += x){
                flag[j] = false;
            }
        }
    }

    for (i = 2; i < MAX; i++){
        if (flag[i]) prime[p++] = i;
    }
}

void init(){
    if (!flag[2]) Sieve();
}

inline int expo(long long x, int n, int m){
    long long res = 1;

    while (n){
        if (n & 1) res = (res * x) % m;
        x = (x * x) % m;
        n >>= 1;
    }

    return (res % m);
}

inline bool miller_rabin(int p){
    if (p < MAX) return flag[p];
    if ((p + 1) & 1) return false;
    for (int i = 1; i < 9; i++){
        if (!(p % prime[i])) return false;
    }

    int a, m, x, s = p - 1, y = p - 1;
    s = s >> __builtin_ctz(s);

    for (int i = 0; i < 3; i++) {
        x = s, a = (base[i] % y) + 1;
        m = expo(a, x, p);
        while ((x != y) && (m != 1) && (m != y)) m = ((long long)m *
m) % p, x <<= 1;
        if ((m != y) && !(x & 1)) return false;
    }
    return true;
```

```
    }
}

int main(){
    prm::init();
}
```

Miller Rabin (Deterministic).cpp

```
#include <bits/stdtr1c++.h>

#define MAX 1000010
#define isprime(x) prm::miller_rabin(x)
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

namespace prm{ /// hash = 130793
    bitset <MAX> flag;
    long double op = 0.0;
    int p = 0, prime[78777];
    const int base[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};

    void Sieve(){
        int i, j, x;
        for (i = 3; i < MAX; i += 2) flag[i] = true;
        for (i = 3, flag[2] = true; (i * i) < MAX; i += 2){
            if (flag[i]){
                for (j = (i * i), x = i << 1; j < MAX; j += x){
                    flag[j] = false;
                }
            }
        }

        for (i = 2; i < MAX; i++){
            if (flag[i]) prime[p++] = i;
        }
    }

    void init(){
        if (!flag[2]) Sieve();
    }

    inline long long mul(long long a, long long b, long long MOD){
        if ((MOD < 3037000500LL)) return ((a * b) % MOD);
        long double res = a;
        res *= b;
        long long c = (long long)(res * op);
        a *= b;
        a -= c * MOD;
        if (a >= MOD) a -= MOD;
        if (a < 0) a += MOD;
```

```
        return a;
    }

    inline long long expo(long long x, long long n, long long m){
        long long res = 1;

        while (n){
            if (n & 1) res = mul(res, x, m);
            x = mul(x, x, m);
            n >>= 1;
        }

        return (res % m);
    }

    inline bool miller_rabin(long long p){
        if (p < MAX) return flag[p];
        if ((p + 1) & 1) return false;
        for (int i = 1; i < 10; i++){ /// basic iterations
            if (!(p % prime[i])) return false;
        }

        long long a, m, x, s = p - 1, y = p - 1;
        op = (long double)1 / p, s = s >> __builtin_ctzll(s);

        for (int i = 0; i < 7; i++) {
            x = s, a = (base[i] % y) + 1;
            m = expo(a, x, p);
            while ((x != y) && (m != 1) && (m != y)) m = mul(m, m, p), x
<<= 1;
            if ((m != y) && !(x & 1)) return false;
        }
        return true;
    }

    inline long long countdiv(long long n){
        int i, j, c;
        long long x, res = 1;
        for (i = 0; i < p; i++){
            x = prime[i];
            if ((x * x * x) > n) break;

            c = 1;
            while (!(n % x)) c++, n /= x;
            res *= c;
        }

        if (miller_rabin(n)) res <<= 1;
        else if (n > 1) {
            x = sqrt((long double)0.95 + n); /// may be change to sqrtl()
?
            if ((x * x) == n && miller_rabin(x)) res *= 3;
            else res <<= 2;
        }
```

```
        return res;
    }
}

int main(){
    prm::init();
}
```

Miller Rabin OP.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>
#include <limits.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

const long long LIM = LONG_LONG_MAX;

long long mul(long long a, long long b, long long m){
    long long x, res;

    if (a < b){
        x = a;
        a = b;
        b = x;
    }
    if (!b) return 0;
    if (a < (LIM / b)) return ((a * b) % m);

    res = 0, x = (a % m);
    while (b){
        if (b & 1){
            res = res + x;
            if (res >= m) res -= m;
        }
        b >>= 1;
        x <<= 1;
        if (x >= m) x -= m;
    }

    return res;
}

long long expo(long long x, long long n, long long m){
    long long res = 1;

    while (n){
        if (n & 1) res = mul(res, x, m);
        x = mul(x, x, m);
```

```
        n >>= 1;
    }

    return (res % m);
}

const int small_primes[] = {3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41,
43, 47, 51, 53, 59, 61, 67, 71};

bool miller_rabin(long long p, int lim){
    if (p < 2) return false;
    if (p == 2) return true;
    if (!(p & 1)) return false;

    int i, val;
    long long a, s, m, x, y;
    for (i = 0; i < 20; i++){
        val = small_primes[i];
        if (p == val) return true;
        if ((p % val) == 0){
            return false;
        }
    }

    srand(time(0));
    s = p - 1, y = p - 1;
    while (!(s & 1)) s >>= 1;

    while (lim--){
        x = s;
        a = (rand() % y) + 1;
        m = expo(a, x, p);

        while ((x != y) && (m != 1) && (m != y)){
            m = mul(m, m, p);
            x <<= 1;
        }
        if ((m != y) && !(x & 1)) return false;
    }

    return true;
}

bool isPrime(long long p){
    if (p < 2) return false;

    long long i;
    for (i = 2; (i * i) <= p; i++){
        if ((p % i) == 0) return false;
    }
    return true;
}

int main(){
```

```
    srand(time(0));
    int i, j, k;
    long long x, y, z;

    int counter = 0;
    for (i = 0; i < 1000; i++){
        long long x = 1LL << 42;
        long long p = x + i;

        int c = isPrime(p);
        int d = miller_rabin(p, 10);

        if (c ^ d){
            counter++;
            printf("%d = %d %d\n", p, c, d);
        }
    }

    printf("counter = %d\n", counter);
    return 0;
}
```

Miller Rabin.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

long long mul(long long a, long long b, long long m){
    long long res = 0;
    long long x = (a % m);

    while (b){
        if (b & 1){
            res = res + x;
            if (res >= m) res -= m;
        }
        b >>= 1;
        x <<= 1;
        if (x >= m) x -= m;
    }

    return res;
}

long long expo(long long x, long long n, long long m){
    long long res = 1;

    while (n){
```

```
        if (n & 1) res = mul(res, x, m);
        x = mul(x, x, m);
        n >>= 1;
    }

    return (res % m);
}

bool miller_rabin(long long p, int lim){
    if (p < 2) return false;
    if (p == 2) return true;
    if (!(p & 1)) return false;

    long long a, s, m, x, y;

    srand(time(0));
    s = p - 1, y = p - 1;
    while (!(s & 1)) s >>= 1;

    while (lim--){
        x = s;
        a = (rand() % y) + 1;
        m = expo(a, x, p);

        while ((x != y) && (m != 1) && (m != y)){
            m = mul(m, m, p);
            x <<= 1;
        }
        if ((m != y) && !(x & 1)) return false;
    }

    return true;
}

bool isPrime(long long p){
    if (p < 2) return false;

    long long i;
    for (i = 2; (i * i) <= p; i++){
        if ((p % i) == 0) return false;
    }
    return true;
}

int main(){
    srand(time(0));
    int i, j, k;
    long long x, y, z;

    int counter = 0;
    for (i = 0; i < 100000; i++){
        int c = isPrime(i);
        int d = miller_rabin(i, 10);
```

```
        if (c ^ d){
            counter++;
            printf("%d = %d %d\n", i, c, d);
        }
    }

    printf("counter = %d\n", counter);
    return 0;
}
```

Miller Rabin.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 1000010
#define isprime(x) prm::miller_rabin(x)
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

namespace prm{
    bitset <MAX> flag;
    long double op = 0.0;
    int p = 0, prime[78777], lim = 15;

    void Sieve(){
        int i, j, x;
        for (i = 3; i < MAX; i += 2) flag[i] = true;
        for (i = 3, flag[2] = true; (i * i) < MAX; i += 2){
            if (flag[i]){
                for (j = (i * i), x = i << 1; j < MAX; j += x){
                    flag[j] = false;
                }
            }
        }

        for (i = 2; i < MAX; i++){
            if (flag[i]) prime[p++] = i;
        }
    }

    void init(){
        if (!flag[2]) Sieve();
    }

    void init(int n){
        lim = n;
        if (!flag[2]) Sieve();
    }

    inline long long mul(long long a, long long b, long long MOD){
       if ((MOD < 3037000500LL)) return ((a * b) % MOD);
```

```
    long double res = a;
    res *= b;
    long long c = (long long)(res * op);
    a *= b;
    a -= c * MOD;
    if (a >= MOD) a -= MOD;
    if (a < 0) a += MOD;
    return a;
  }

  inline long long expo(long long x, long long n, long long m){
      long long res = 1;

      while (n){
          if (n & 1) res = mul(res, x, m);
          x = mul(x, x, m);
          n >>= 1;
      }

      return (res % m);
  }

  inline bool miller_rabin(long long p){
      if (p < MAX) return flag[p];
      if ((p + 1) & 1) return false;
      for (int i = 1; i < 10; i++){
          if (!(p % prime[i])) return false;
      }

      long long a, m, x, s = p - 1, y = p - 1;
      op = (long double)1 / p, s = s >> __builtin_ctzll(s);

      for (int i = 0; i < lim; i++) {
          x = s, a = (rand() % y) + 1;
          m = expo(a, x, p);
          while ((x != y) && (m != 1) && (m != y)) m = mul(m, m, p), x
<<= 1;
          if ((m != y) && !(x & 1)) return false;
      }
      return true;
  }

  inline long long countdiv(long long n){
      int i, j, c;
      long long x, res = 1;
      for (i = 0; i < p; i++){
          x = prime[i];
          if ((x * x * x) > n) break;

          c = 1;
          while (!(n % x)) c++, n /= x;
          res *= c;
      }
```

```
        if (miller_rabin(n)) res <<= 1;
        else if (n > 1) {
            x = sqrt((long double)0.95 + n); ///
            if ((x * x) == n && miller_rabin(x)) res *= 3;
            else res <<= 2;
        }

        return res;
    }
}

int main(){
    long long n;
    prm::init();

    while (scanf("%lld", &n) != EOF){
        printf("%lld\n", prm::countdiv(n));
    }
    return 0;
}
```

Mincost Maxflow (Bellman Ford).cpp

```
#include <stdio.h>
#include <bits/stdtr1c++.h>

#define MAX 200010 /// 2 * max(nodes, edges)
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Min-cost Max-flow using bellman ford
/// 0 Based indexed for directed weighted graphs (for undirected graphs,
just add two directed edges)
/// Runs in around 0.75 seconds when n <= 100 and m = n * (n - 1) / 2
/// Runs well for sparse graphs though, e.g, m <= 10 * n
/// Costs must be non-negative

namespace mcmf{
    const long long INF = 1LL << 60;

    long long dis[MAX], cap[MAX], cost[MAX];
    int n, m, s, t, to[MAX], from[MAX], last[MAX], next[MAX], adj[MAX];

    void init(int nodes, int source, int sink){
        m = 0, n = nodes;
        s = source, t = sink;
        for (int i = 0; i <= n; i++) last[i] = -1;
    }

    void addEdge(int u, int v, long long c, long long w){
```

```
        from[m] = u, adj[m] = v, cap[m] = c, cost[m] = w, next[m] =
last[u], last[u] = m++;
        from[m] = v, adj[m] = u, cap[m] = 0, cost[m] = -w, next[m] =
last[v], last[v] = m++;
    }

    bool bellman_ford(){
        int i, u, v, e, flag = 1;
        for (i = 0; i < n; i++) to[i] = -1;
        for (i = 0; i < n; i++) dis[i] = INF;

        dis[s] = 0;
        for (i = 0; i < n && flag; i++){
            for (u = 0, flag = 0; u < n; u++){
                for (e = last[u]; e != -1; e = next[e]){
                    v = adj[e];
                    if (cap[e] && dis[v] > (dis[u] + cost[e])){
                        dis[v] = dis[u] + cost[e];
                        to[v] = e;
                        flag = 1;
                    }
                }
            }
        }
        ///assert(i < n); /// Negative cycle found
        return (dis[t] < INF);
    }

    pair<long long, long long> solve(){
        long long maxflow = 0, mincost = 0;

        while (bellman_ford()){
            long long aug = INF;
            for (int e = to[t]; e != -1; e = to[from[e]]) aug = min(aug,
cap[e]);
            for (int e = to[t]; e != -1; e = to[from[e]]){
                mincost += (aug * cost[e]);
                cap[e] -= aug, cap[e ^ 1] += aug;
            }
            maxflow += aug;
        }
        return make_pair(mincost, maxflow);
    }
}

int main(){

}
```

Mincost Maxflow (Dijkstra + Potentials).cpp

```
-------------------------------------------------
#include <stdio.h>
#include <bits/stdtr1c++.h>
```

```
#define MAX 200010 /// 2 * max(nodes, edges)
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Min-cost Max-flow using dijkstra with potentials
/// 0 Based indexed for directed weighted graphs (for undirected graphs,
just add two directed edges)
/// Runs in around 2 seconds when n <= 200 and m = n * (n - 1) / 2
/// Runs well for sparse graphs though, e.g, m <= 10 * n
/// Costs must be non-negative

namespace mcmf{
   const long long INF = 1LL << 60;

   long long potential[MAX], dis[MAX], cap[MAX], cost[MAX];
   int n, m, s, t, to[MAX], from[MAX], last[MAX], next[MAX], adj[MAX];

   struct compare{
       inline bool operator()(int a, int b){
           if (dis[a] == dis[b]) return (a < b);
           return (dis[a] < dis[b]);
       }
   };
   set<int, compare> S;

   void init(int nodes, int source, int sink){
       m = 0, n = nodes;
       s = source, t = sink;
       for (int i = 0; i <= n; i++) potential[i] = 0, last[i] = -1;
   }

   void addEdge(int u, int v, long long c, long long w){
       from[m] = u, adj[m] = v, cap[m] = c, cost[m] = w, next[m] =
last[u], last[u] = m++;
       from[m] = v, adj[m] = u, cap[m] = 0, cost[m] = -w, next[m] =
last[v], last[v] = m++;
   }

   pair<long long, long long> solve(){
       int i, j, e, u, v;
       long long w, aug = 0, mincost = 0, maxflow = 0;

       while (1){
           S.clear();
           for (i = 0; i < n; i++) dis[i] = INF;

           dis[s] = 0, S.insert(s);
           while (!S.empty()){
               u = *(S.begin());
               if (u == t) break;
```

```
                S.erase(S.begin());

                for (e = last[u]; e != -1; e = next[e]){
                    if (cap[e] != 0){
                        v = adj[e];
                        w = dis[u] + potential[u] + cost[e] -
potential[v];

                        if (dis[v] > w){
                            S.erase(v);
                            dis[v] = w, to[v] = e;
                            S.insert(v);
                        }
                    }
                }
            }
            if (dis[t] >= (INF >> 1)) break;

            aug = cap[to[t]];
            for (i = t; i != s; i = from[to[i]]) aug = min(aug,
cap[to[i]]);
            for (i = t; i != s; i = from[to[i]]){
                cap[to[i]] -= aug;
                cap[to[i] ^ 1] += aug;
                mincost += (cost[to[i]] * aug);
            }
            for (i = 0; i <= n; i++) potential[i] = min(potential[i] +
dis[i], INF);
            maxflow += aug;
        }
        return make_pair(mincost, maxflow);
    }
}

int main(){

}
```

Mincost Maxflow (Network Simplex).cpp

```
--------------------------------------------------
#include <stdio.h>
#include <bits/stdtr1c++.h>

#define MAXV 12010
#define MAXC 12210

#define EQUAL 0
#define LESSEQ -1
#define GREATEQ 1
#define MINIMIZE -1
#define MAXIMIZE +1

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
```

```
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Min-cost Max-flow using network simplex
/// 0 Based indexed for directed weighted graphs (for undirected graphs,
just add two directed edges)
/// Runs in around 1.25 seconds when n <= 100 and m = n * (n - 1) / 2
/// Costs can be negative and works with negative cycles as well
/// Number of variables in simplex = M, Number of constraints = N + M

struct edge{
   int u, v;
   long long c, w; /// c = capacity, w = weight of edge
   edge(){}
   edge(int a, int b, long long x, long long y){
       u = a, v = b;
       c = x, w = y;
   }
};

namespace lp{
   bool has_solution;
   int n, m, flag, link[MAXC], down[MAXV], idx[MAXV];
   long long res, ar[MAXC][MAXV], val[MAXV], rhs[MAXC];

   void init(int nvar, long long func[], int min_or_max){ /// func[] =
objective function
       flag = min_or_max;
       has_solution = false;
       res = 0, m = 0, n = nvar;
       for (int i = 1; i <= n; i++) idx[i] = 0;
       for (int i = 1; i <= n; i++) ar[0][i] = func[i] * flag;
   }

   /// var[] = co-efficients of the constraints (LHS), lim = limit in RHS
   inline void add_constraint(long long var[], int lim, int flag){
       flag *= -1;
       if (flag == 0){
           rhs[++m] = lim;
           for (int i = 1; i <= n; i++) ar[m][i] = var[i];
           rhs[++m] = -lim;
           for (int i = 1; i <= n; i++) ar[m][i] = -var[i];
       }
       else{
           rhs[++m] = lim * flag;
           for (int i = 1; i <= n; i++) ar[m][i] = var[i] * flag;
       }
   }

   void pivot(int x, int y){ /// pivoting and exchanging a non-basic
variable with a basic variable
           int i, j, t = 0;
```

```
        long long v = ar[x][y];

        rhs[x] /= v;
        ar[x][y] = 1;
        swap(link[x], down[y]);
            for (j = 1; j <= n; j++){
            ar[x][j] /= v;
            if (ar[x][j]) val[++t] = j;
            }

            for (i = 1; i <= m; i++){
            if (ar[i][y] && i != x){
                rhs[i] -= (ar[i][y] * rhs[x]);
                v = ar[i][y], ar[i][y] = 0;
                for (j = 1; j <= t; j++) ar[i][val[j]] -= (v *
ar[x][val[j]]);
            }
            }

            res += (ar[0][y] * rhs[x]), v = ar[0][y], ar[0][y] = 0;
            for (j = 1; j <= t; j++) ar[0][val[j]] -= (v *
ar[x][val[j]]);
    }

    long long solve(){ /// simplex core
        int i, j, x, y;
        for (i = 1; i <= n; i++) down[i] = i;
        for (i = 1; i <= m; i++) link[i] = i + n;

        while (1){
            for (x = 1; x <= m && rhs[x] >= 0; x++){}
            if (x > m) break;
            for (j = 1, y = 0; j <= n; j++){
                if (ar[x][j] < 0){
                    y = j;
                    if (rand() & 1) break; /// try removing rand()
                }
            }
            if (y == 0) return -666;
            pivot(x, y);
        }

        while (1){
            for (i = 1, y = 0; i <= n; i++){
                if (ar[0][i] > 0 && (y == 0 || ar[0][i] > ar[0][y])) y =
i;
            }
            if (y == 0) break;
            for (j = 1, x = 0; j <= m; j++){
                if (ar[j][y] > 0){
                    if (x == 0 || rhs[j] / ar[j][y] < rhs[x] / ar[x][y]) x
= j;
                }
            }
```

```
            assert(x != 0);
            pivot(x, y);
        }

        has_solution = true;
        for (int i = 1; i <= m; i++){
            if(link[i] <= n) idx[link[i]] = i;
        }
        for (int i = 1; i <= n; i++) val[i] = rhs[idx[i]];
            return res * flag;
      }
}

void buildFlowNetwork(int n, int s, int t, vector <edge> E, int flag,
long long flow = 0){
   long long ar[MAXV] = {0};
   vector <pair<int, int>> in[n + 1], out[n + 1];

   for (int i = 0; i < E.size(); i++){
       if (E[i].u == t || E[i].v == s) continue;
       out[E[i].u].push_back(make_pair(i + 1, E[i].c));
       in[E[i].v].push_back(make_pair(i + 1, E[i].c));
   }

   clr(ar);
   if (flag == 0){ /// to find maximum flow
       for (int i = 0; i < out[s].size(); i++) ar[out[s][i].first] += 1;
       lp::init(E.size(), ar, MAXIMIZE);
   }
   else{ /// to find minimum cost
       for (int i = 0; i < E.size(); i++) ar[i + 1] += E[i].w;
       lp::init(E.size(), ar, MINIMIZE);
   }

   if (flag == 1){ /// add additional constraint that total flow from
source equals maximum flow
       clr(ar);
       for (int i = 0; i < out[s].size(); i++) ar[out[s][i].first] += 1;
       lp::add_constraint(ar, flow, EQUAL);
   }

   clr(ar);
   for (int i = 0; i < out[s].size(); i++) ar[out[s][i].first] += 1;
   for (int i = 0; i < in[t].size(); i++) ar[in[t][i].first] += -1;
   lp::add_constraint(ar, 0, LESSEQ);

   clr(ar);
   for (int i = 0; i < E.size(); i++){
       ar[i + 1] = 1;
       lp::add_constraint(ar, E[i].c, LESSEQ);
       ar[i + 1] = 0;
   }

   for (int i = 0; i < n; i++){
```

```
        if (i != s && i != t){
            clr(ar);
            for (int j = 0; j < out[i].size(); j++) ar[out[i][j].first] +=
1;
            for (int j = 0; j < in[i].size(); j++) ar[in[i][j].first] += -
1;
            lp::add_constraint(ar, 0, LESSEQ);
        }
    }
}

/// n = nodes, s = source, t = sink, E = list of edges
pair<long long, long long> solve(int n, int s, int t, vector <edge> E){
    long long flow = 0, cost = 0;
    buildFlowNetwork(n, s, t, E, 0);

    flow = lp::solve();
    if (flow > 0){
        buildFlowNetwork(n, s, t, E, 1, flow);
        cost = lp::solve();
    }
    return make_pair(cost, flow);
}

int main(){

}
```

Minimum Lexicographic Rotation.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

/// Lexicographically Minimum String Rotation
int minlex(char* str){ /// Returns the 0-based index
    int i, j, k, n, len, x, y;
    len = n = strlen(str), n <<= 1, i = 0, j = 1, k = 0;

    while((i + k) < n && (j + k) < n) {
        x = i + k >= len ? str[i + k - len] : str[i + k];
        y = j + k >= len ? str[j + k - len] : str[j + k];
        if(x == y) k++;
        else if (x < y){
            j += ++k, k = 0;
            if (i >= j) j = i + 1;
        }
        else{
            i += ++k, k = 0;
            if (j >= i) i = j + 1;
        }
```

```
    }

    return (i < j) ? i : j;
}

int t;
char str[50010];

int main(){
    gets(str);
    while (gets(str)){
        printf("%d\n", minlex(str));
    }
    return 0;
}
```

Minimum Path Cover in DAG.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 505
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Minimum path cover/Maximum independent set in DAG
namespace dag{
    bool ar[MAX][MAX]; /// For transitive closure and minimum path cover
with not necessarily disjoint vertex

    vector <int> adj[MAX];
    bool visited[MAX], first_set[MAX], second_set[MAX];
    int n, L[MAX], R[MAX], D[MAX], Q[MAX], dis[MAX], parent[MAX];

    inline void init(int nodes){ /// Number of vertices in DAG
        n = nodes;
        for (int i = 0; i < MAX; i++) adj[i].clear();
    }

    inline void add_edge(int u, int v){ /// 0 based index, directed edge
of DAG
        adj[u].push_back(v);
    }

    bool dfs(int i){
        int len = adj[i].size();
        for (int j = 0; j < len; j++){
            int x = adj[i][j];
            if (L[x] == -1 || (parent[L[x]] == i)){
                if (L[x] == -1 || dfs(L[x])){
                    L[x] = i, R[i] = x;
```

```
                return true;
            }
        }
    }
    return false;
}

bool bfs(){
    clr(visited);
    int i, j, x, d, f = 0, l = 0;

    for (i = 0; i < n; i++){
        if (R[i] == -1){
            visited[i] = true;
            Q[l++] = i, dis[i] = 0;
        }
    }

    while (f < l){
        i = Q[f++];
        int len = adj[i].size();
        for (j = 0; j < len; j++){
            x = adj[i][j], d = L[x];
            if (d == -1) return true;

            else if (!visited[d]){
                Q[l++] = d;
                parent[d] = i, visited[d] = true, dis[d] = dis[i] + 1;
            }
        }
    }
    return false;
}

void get_path(int i){
    first_set[i] = true;
    int j, x, len = adj[i].size();

    for (j = 0; j < len; j++){
        x = adj[i][j];
        if (!second_set[x] && L[x] != -1){
            second_set[x] = true;
            get_path(L[x]);
        }
    }
}

void transitive_closure(){ /// Transitive closure in O(n * m)
    clr(ar);
    int i, j, k, l;
    for (i = 0; i < n; i++){
        l = adj[i].size();
        for (j = 0; j < l; j++){
            ar[i][adj[i][j]] = true;
```

```
            }
            adj[i].clear();
        }

        for (k = 0; k < n; k++){
            for (i = 0; i < n; i++){
                if (ar[i][k]){
                    for (j = 0; j < n; j++){
                        if (ar[k][j]) ar[i][j] = true;
                    }
                }
            }
        }

        for (i = 0; i < n; i++){
            for (j = 0; j < n; j++){
                if (i != j && ar[i][j]){
                    adj[i].push_back(j);
                }
            }
        }
    }

    int minimum_disjoint_path_cover(){ /// Minimum vertex disjoint path
cover in DAG. Handle isolated vertices appropriately
        int i, res = 0;
        memset(L, -1, sizeof(L));
        memset(R, -1, sizeof(R));

        while (bfs()){
            for (i = 0; i < n; i++){
                if (R[i] == -1 && dfs(i)) res++;
            }
        }

        return n - res;
    }

    int minimum_path_cover(){ /// Minimum path cover in DAG. Handle
isolated vertices appropriately
        transitive_closure();
        return minimum_disjoint_path_cover();
    }

    vector <int> minimum_vertex_cover(){ /// Minimum vertex cover of DAG,
equal to maximum bipartite matching
        int i, res = 0;
        memset(L, -1, sizeof(L));
        memset(R, -1, sizeof(R));

        while (bfs()){
            for (i = 0; i < n; i++){
                if (R[i] == -1 && dfs(i)) res++;
            }
```

```
        }

        vector <int> v;
        clr(first_set), clr(second_set);
        for (i = 0; i < n; i++){
            if (R[i] == -1) get_path(i);
        }

        for (i = 0; i < n; i++){
            if (!first_set[i] || second_set[i]) v.push_back(i);
        }

        return v;
    }

    vector <int> maximum_independent_set(){ /// Maximum independent set of
DAG, all vertices not in minimum vertex cover
        vector <int> v = minimum_vertex_cover();
        clr(visited);
        int i, len = v.size();
        for (i = 0; i < len; i++) visited[v[i]] = true;

        vector <int> res;
        for (i = 0; i < n; i++){
            if (!visited[i]) res.push_back(i);
        }
        return res;
    }
}

int main(){
}

Misc.c
--------------------------------------------------
#include <stdio.h>

/***

1. Sum of all numbers less than or equal to n and also co-prime to n =
(phi(n) * n) / 2

2. Lagrange's Polnomial:
  A polynomial of degree n can be uniquely represented and evaluated from
it's value in n + 1 distinct points (x_i, y_i)

  For the function f(x) = x ^2, given three points

   x_0 = 1, f(x_0) = 1
   x_1 = 2, f(x_1) = 4
   x_2 = 3, f(x_2) = 9

   The interpolating polynomial is:
```

```
            1*(x-2)(x-3) + 4*(x-1)(x-3) + 9*(x-1)(x-2)
   L(x) =    ----------     ----------     ----------    =  x^2
              (1-2)(1-3)     (2-1)(2-3)     (3-1)(3-2)

3. Eulerian Numbers, E(n, k): Number of permutations from 1 to n such
that
  P_i < P_(i+1) (or P_i > P_(i + 1) since symmetric) in exactly k
positions.
  Recurrence: E(n, k) = (k+1)E(n - 1, k) + (n - k)(n - 1, k - 1)
***/

int main(){
}
```

Mo_s Algorithm 2.cpp

```
#include <bits/stdtr1c++.h>

#define MAXN 200010
#define MAXQ 200010
#define MAXV 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

const int block_size = 533;

namespace mo{ /// 0 based index for arrays and queries
   long long res, out[MAXQ];
   int q, lim, ar[MAXN], freq[MAXV];

   struct query{
       int l, r, d, idx;

       inline query() {}
       inline query(int a, int b, int c){
           idx = c;
           l = a, r = b, d = l / block_size;
       }

       inline bool operator < (const query& other) const{
           if (d != other.d) return (d < other.d);
           if (r == other.r) return (l < other.l);
           return (r < other.r);
       }
   } Q[MAXQ];

   inline void init(int n, int *T){
       clr(freq);
       res = 0, q = 0;
       for (int i = 0; i < n; i++) ar[i] = T[i];
```

```
        lim = (*max_element(ar, ar + n) + 1) * sizeof(int); /// long long
freq
    }

    inline void push(int a, int b){
        Q[q] = query(a, b, q);
        q++;
    }

    inline void insert(int idx){
        res -= (long long)ar[idx] * freq[ar[idx]] * freq[ar[idx]];
        freq[ar[idx]]++;
        res += (long long)ar[idx] * freq[ar[idx]] * freq[ar[idx]];
    }

    inline void erase(int idx){
        res -= (long long)ar[idx] * freq[ar[idx]] * freq[ar[idx]];
        freq[ar[idx]]--;
        res += (long long)ar[idx] * freq[ar[idx]] * freq[ar[idx]];
    }

    inline void run(){
        sort(Q, Q + q);
        int i, j, l, r, a = 0, b = MAXN;

        for (i = 0; i < q; i++){
            l = Q[i].l, r = Q[i].r;
            if (b > r){
                res = 0;
                memset(freq, 0, lim);
                for (j = l; j <= r; j++) insert(j);
            }
            else{
                for (j = l; j < a; j++) insert(j);
                for (j = b + 1; j <= r; j++) insert(j);
                for (j = a; j < l; j++) erase(j);
                for (j = r + 1; j <= b; j++) erase(j);
            }

            a = l, b = r;
            out[Q[i].idx] = res;
        }
        for (i = 0; i < q; i++) printf("%lld\n", out[i]);
    }
}

int n, ar[MAXN];

int main(){
    int n, q, i, j, k, a, b;

    scanf("%d %d", &n, &q);
    for (i = 0; i < n; i++) scanf("%d", &ar[i]);
```

```
    mo::init(n, ar);
    while (q--){
        scanf("%d %d", &a, &b);
        mo::push(a - 1, b - 1);
    }
    mo::run();
    return 0;
}
```

Mo_s Algorithm 3.cpp

```
#include <bits/stdtr1c++.h>

#define MAXN 200010
#define MAXQ 200010
#define MAXV 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

using namespace std;

/// 0 based index for arrays and queries

const int block_size = 633;

long long res, out[MAXQ];
int n, q, ar[MAXN], freq[MAXV];

struct query{
    int l, r, d, i;

    inline query() {}
    inline query(int a, int b, int c){
        i = c;
        l = a, r = b, d = l / block_size;
    }

    inline bool operator < (const query& other) const{
        if (d != other.d) return (d < other.d);
        return ((d & 1) ? (r < other.r) : (r > other.r));
    }
} Q[MAXQ];

inline void insert(int i){
    res += (long long)ar[i] * (1 + 2 * freq[ar[i]]++);
}

inline void erase(int i){
    res -= (long long)ar[i] * (1 + 2 * --freq[ar[i]]);
}

inline void run(){ /// hash = 812195
    sort(Q, Q + q);
    int i, l, r, a = 0, b = 0;
```

```
    for (res = 0, i = 0; i < q; i++){
        l = Q[i].l, r = Q[i].r;
        while (a > l) insert(--a);
        while (b <= r) insert(b++);
        while (a < l) erase(a++);
        while (b > (r + 1)) erase(--b);
        out[Q[i].i] = res;
    }
    for (i = 0; i < q; i++) printf("%lld\n", out[i]);
}

int main(){
    int n, i, j, k, a, b;

    scanf("%d %d", &n, &q);
    for (i = 0; i < n; i++) scanf("%d", &ar[i]);
    for (i = 0; i < q; i++){
        scanf("%d %d", &a, &b);
        Q[i] = query(a - 1, b - 1, i);
    }

    run();
    return 0;
}
```

Mo_s Algorithm On Trees (Edges).cpp

```
--------------------------------------------------
/// Frank Sinatra, Winter Training Camp Moscow SU Trinity Contest
/// Given a weighted tree, find the minimum non-negative value which does not occur in the path from u to v

#include <bits/stdtr1c++.h>

#define MAXN 100010
#define MAXQ 100010
#define MAXV 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

const int block_size = 633;
typedef pair<int, int> Pair;

namespace mo{ /// Mo's Algorithm on tree(for nodes), 1-based index for nodes and queries
    char visited[MAXN];
    vector <Pair> adj[MAXN];
    int t, q, n, out[MAXQ], dp[MAXV], freq[MAXV], val[MAXN], depth[MAXN], parent[MAXN], discover[MAXN];
```

```
struct query{
    int l, r, idx;

    inline query(){}
    inline query(int a, int b, int c){
        idx = c;
        l = a, r = b;
    }

    inline bool operator < (const query& other) const{
        int d1 = discover[l] / block_size, d2 = discover[other.l] /
block_size;
        if (d1 != d2) return (d1 < d2);
        return ((d1 & 1) ? (discover[r] < discover[other.r]) :
(discover[r] > discover[other.r])); /// experiment
    }
} Q[MAXQ];

void init(int nodes){
    t = q = 0, n = nodes;
    for (int i = 0; i < MAXN; i++) adj[i].clear();
}

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

inline void push(int l, int r){
    q++;
    Q[q] = query(l, r, q);
}

inline void dfs(int i){
    discover[i] = ++t;
    int j, x, len = adj[i].size();
    for (j = 0; j < len; j++){
        x = adj[i][j].first;
        if (x != parent[i]){
            val[x] = adj[i][j].second;
            parent[x] = i, depth[x] = depth[i] + 1;
            dfs(x);
        }
    }
}

inline void jump(int& i){
    if (!visited[i]){
        if (!freq[val[i]]++) dp[val[i] / block_size]++; /// insert
    }
    else{
        if (!--freq[val[i]]) dp[val[i] / block_size]--; /// delete
    }
    visited[i] ^= 1;
```

```
        i = parent[i];
    }

    inline void update(int u, int v){
        while (depth[u] > depth[v]) jump(u);
        while (depth[u] < depth[v]) jump(v);
        while (u != v) jump(u), jump(v);
    }

    inline void run(){
        clr(dp), clr(freq), clr(visited);
        parent[1] = 1, depth[1] = 0, Q[0] = query(1, 1, 0);

        dfs(1);
        sort(Q + 1, Q + q + 1);
        for (int i = 1; i <= q; i++){
            update(Q[i - 1].l, Q[i].l);
            update(Q[i - 1].r, Q[i].r);
            int res = 0, pos = 0;
            while (dp[pos] == block_size) res += block_size, pos++;
            while (freq[res] > 0) res++;
            out[Q[i].idx] = res;
        }
        for (int i = 1; i <= q; i++) printf("%d\n", out[i]);
    }
}

int main(){
    int n, q, i, j, k, a, b, c;

    scanf("%d %d", &n, &q);
    mo::init(n);
    for (i = 1; i < n; i++){
        scanf("%d %d %d", &a, &b, &c);
        c = min(c, n + 1); /// Only for this problem since weights can be
up to 10^9
        mo::add_edge(a, b, c);
    }

    for (i = 1; i <= q; i++){
        scanf("%d %d", &a, &b);
        mo::push(a, b);
    }
    mo::run();
    return 0;
}
```

Mo_s Algorithm On Trees (Nodes).cpp

```
--------------------------------------------------
/// SPOJ Count On A Tree II, Number of distinct values on nodes on the
path from U to V

#include <bits/stdtr1c++.h>
```

```
#define LOGN 18
#define MAXN 100010
#define MAXQ 100010
#define MAXV 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

const int block_size = 341;

namespace mo{ /// Mo's Algorithm on tree(for nodes), 1-based index for
nodes and queries
   char visited[MAXN];
   vector <int> adj[MAXN];
   int t, q, n, res, out[MAXQ], freq[MAXV], lg[MAXN], val[MAXN],
depth[MAXN], parent[MAXN], discover[MAXN], dp[MAXN][LOGN];

   struct query{
        int l, r, idx;

        inline query(){}
        inline query(int a, int b, int c){
            idx = c;
            l = a, r = b;
        }

        inline bool operator < (const query& other) const{
            int d1 = discover[l] / block_size, d2 = discover[other.l] /
block_size;
            if (d1 != d2) return (d1 < d2);
            return ((d1 & 1) ? (discover[r] < discover[other.r]) :
(discover[r] > discover[other.r])); /// experiment
        }
   } Q[MAXQ];

   void init(int nodes, int nodeval[MAXN]){
        t = q = res = 0, n = nodes;
        for (int i = 0; i < MAXN; i++) adj[i].clear();
        for (int i = 1; i <= nodes; i++) val[i] = nodeval[i];
   }

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

   inline void push(int l, int r, int idx){
        Q[++q] = query(l, r, idx);
   }

   inline void dfs(int i){
```

```
    discover[i] = ++t;
    int j, x, len = adj[i].size();
    for (j = 0; j < len; j++){
        x = adj[i][j];
        if (x != parent[i]){
            parent[x] = i, depth[x] = depth[i] + 1;
            dfs(x);
        }
    }
}

inline int lca(int a, int b){
    if (a == b) return a;
    if (depth[a] < depth[b]) swap(a, b);

    for (int i = lg[depth[a] - depth[b]]; i >= 0; i--){
        if ((depth[a] - (1 << i)) >= depth[b]) a = dp[a][i];
    }
    if (a == b) return a;

    for (int i = lg[depth[a]]; i >= 0; i--){
        if (dp[a][i] != dp[b][i]){
            a = dp[a][i];
            b = dp[b][i];
        }
    }

    return (a == b) ? a : parent[a];
}

inline void build(){
    clr(freq), clr(visited);
    parent[1] = 1, depth[1] = 0, lg[0] = lg[1] = 0;
    for (int i = 2; i <= n; i++) lg[i] = lg[i >> 1] + 1;

    dfs(1);
    for (int l = 0; l <= lg[n]; l++){
        for (int i = 1; i <= n; i++){
            if (!l) dp[i][l] = parent[i];
            else dp[i][l] = dp[dp[i][l - 1]][l - 1];
        }
    }
}

inline void update(int i){
    if (!visited[i]){
        if (!freq[val[i]]++) res++; /// insert
    }
    else{
        if (!--freq[val[i]]) res--; /// delete
    }
    visited[i] ^= 1;
}
```

```
    inline void update_path(int u, int v){
        while (depth[u] > depth[v]) update(u), u = parent[u];
        while (depth[u] < depth[v]) update(v), v = parent[v];
        while (u != v){
            update(u), update(v);
            u = parent[u], v = parent[v];
        }
        update(u);
    }

    inline void run(){
        build();
        update(1);

        Q[0] = query(1, 1, 0);
        sort(Q + 1, Q + q + 1);

        for (int i = 1; i <= q; i++){
            update_path(Q[i - 1].l, Q[i].l);
            update_path(Q[i - 1].r, Q[i].r);
            int l1 = lca(Q[i - 1].r, Q[i].r), l2 = lca(Q[i - 1].l,
Q[i].l);
            int l3 = lca(Q[i].l, Q[i].r), l4 = lca(Q[i - 1].l, Q[i -
1].r);
            update(l1), update(l2), update(l3), update(l4);
            out[Q[i].idx] = res;
        }
        for (int i = 1; i <= q; i++) printf("%d\n", out[i]);
    }
}

int T[MAXN], H[MAXN];
tr1::unordered_map <int, int> mp;

int main(){
    int n, q, h, i, j, k, a, b;

    scanf("%d %d", &n, &q);
    for (i = 1; i <= n; i++) scanf("%d", &T[i]);

    ///Co-ordinate compression, necessary only for this problem

    /*** mp.clear();
    for (h = 0, i = 0; i < n; i++) H[i] = T[i + 1];
    sort(H, H + n);
    H[n] = -1;
    for (i = 0; i < n; i++){
        if (H[i] != H[i + 1]) mp[H[i]] = ++h;
    }
    for (i = 1; i <= n; i++) T[i] = mp[T[i]]; ***/

    mo::init(n, T);
    for (i = 1; i < n; i++){
        scanf("%d %d", &a, &b);
```

```
        mo::add_edge(a, b);
    }

    for (i = 1; i <= q; i++) {
        scanf("%d %d", &a, &b);
        mo::push(a, b, i);
    }

    mo::run();
    return 0;
}
```

Mo_s Algorithm On Trees.cpp

```
#include <bits/stdtr1c++.h>

#define LOG 18
#define MAX 100010
#define MAXQ 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// SPOJ Count On A Tree II, Number of distinct values on nodes on the
path from U to V

namespace mo{ /// Mo's Algorithm on Trees, 0-based index
    int q, counter, block_size = 337, out[MAXQ], freq[MAX]; /// block_size
= sqrt(2*N)

    struct Query{
        int l, r, s, d, idx;

        inline Query() {}
        inline Query(int a, int b, int p, int c){
            idx = c;
            l = a, r = b, s = p, d = l / block_size;
        }

        inline bool operator < (const Query& other) const{
            if (d != other.d) return (d < other.d);
            return ((d & 1) ? (r < other.r) : (r > other.r));
        }
    } Q[MAXQ];

    vector <int> adj[MAX];
    int n, r, t, S[MAX], E[MAX], ar[MAX << 1], parent[MAX], depth[MAX],
lg[MAX], nodeval[MAX], nodecount[MAX], dp[MAX][LOG];

    void init(int nodes, int root, int sz, int* nodevalT){ /// Values on
nodes, modify appropriately if values on edges
```

```
    lg[0] = lg[1] = 0;
    block_size = max(1, sz);
    clr(freq), clr(nodecount);
    q = 0, t = 0, n = nodes, r = root;
    for (int i = 0; i <= n; i++) adj[i].clear();
    for (int i = 2; i <= n; i++) lg[i] = lg[i >> 1] + 1;
    for (int i = 0; i < n; i++) nodeval[i] = nodevalT[i];
}

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

inline int lca(int a, int b){
    if (a == b) return a;
    if (depth[a] < depth[b]) swap(a, b);

    for (int i = lg[depth[a] - depth[b]]; i >= 0; i--){
        if ((depth[a] - (1 << i)) >= depth[b]) a = dp[a][i];
    }
    if (a == b) return a;

    for (int i = lg[depth[a]]; i >= 0; i--){
        if (dp[a][i] != dp[b][i]){
            a = dp[a][i];
            b = dp[b][i];
        }
    }

    return (a == b) ? a : parent[a];
}

inline void dfs(int i, int p){
    S[i] = t, ar[t++] = i;
    int j, len = adj[i].size();
    for (j = 0, parent[i] = p; j < len; j++){
        if (adj[i][j] != p){
            depth[adj[i][j]] = depth[i] + 1;
            dfs(adj[i][j], i);
        }
    }
    E[i] = t, ar[t++] = i;
}

inline void build(){
    depth[r] = 0;
    dfs(r, r);

    for (int l = 0; l <= lg[n]; l++){
        for (int i = 0; i < n; i++){
            if (!l) dp[i][l] = parent[i];
            else dp[i][l] = dp[dp[i][l - 1]][l - 1];
        }
```

```
        }
    }

    inline void push(int a, int b, int idx){
        if (depth[a] > depth[b]) swap(a, b);
        int l = lca(a, b);
        if (a == l) Q[q++] = Query(S[a], S[b], -1, idx);
        else{
            if (E[b] <= S[a]) Q[q++] = Query(E[b], S[a], S[l], idx);
            else Q[q++] = Query(E[a], S[b], S[l], idx);
        }
    }

    /// If a node occurs twice in a range, then both its value needs to be
ignored
    inline void insert(int idx){
        int x = ar[idx];
        if (nodecount[x]){
            freq[nodeval[x]]--;
            if (freq[nodeval[x]] == 0) counter--;
        }
        else{
            if (freq[nodeval[x]] == 0) counter++;
            freq[nodeval[x]]++;
        }
        nodecount[x] ^= 1;
    }

    inline void erase(int idx){
        int x = ar[idx];
        if (!nodecount[x]){
            if (freq[nodeval[x]] == 0) counter++;
            freq[nodeval[x]]++;
        }
        else{
            freq[nodeval[x]]--;
            if (freq[nodeval[x]] == 0) counter--;
        }
        nodecount[x] ^= 1;
    }

    inline void run(){
        counter = 0;
        sort(Q, Q + q);
        int i, l, r, a = 0, b = 0; /// Change here if 1-based array

        for (i = 0; i < q; i++){
            l = Q[i].l, r = Q[i].r;
            while (a > l) insert(--a);
            while (b <= r) insert(b++);
            while (a < l) erase(a++);
            while (b > (r + 1)) erase(--b);
            if (Q[i].s != -1) insert(Q[i].s);
            out[Q[i].idx] = counter;
```

```
            if (Q[i].s != -1) erase(Q[i].s);
        }
    }

    inline void print(){
        for (int i = 0; i < q; i++) printf("%d\n", out[i]);
    }
}

int T[MAX], H[MAX];
tr1::unordered_map <int, int> mp;

int main(){
    int n, q, h, i, j, k, a, b;

    scanf("%d %d", &n, &q);
    for (i = 0; i < n; i++) scanf("%d", &T[i]);

    /// Co-ordinate compression, necessary only for this problem
    /*** mp.clear();
    for (h = 0, i = 0; i < n; i++) H[i] = T[i];
    sort(H, H + n);
    H[n] = -1;
    for (i = 0; i < n; i++){
        if (H[i] != H[i + 1]) mp[H[i]] = ++h;
    }
    for (i = 0; i < n; i++) T[i] = mp[T[i]]; ***/

    mo::init(n, 0, 337, T);
    for (i = 1; i < n; i++){
        scanf("%d %d", &a, &b);
        a--, b--;
        mo::add_edge(a, b);
    }
    mo::build();

    for (i = 0; i < q; i++) {
        scanf("%d %d", &a, &b);
        a--, b--;
        mo::push(a, b, i);
    }

    mo::run();
    mo::print();
    return 0;
}
```

Mo_s Algorithm.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
```

```
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/// Mo's Algorithm for DQUERY, 0-based index by default
namespace mo{
   int q, counter, block_size = 437, out[MAX], ar[MAX], freq[MAX];

   struct Query{
       int l, r, d, idx;

       inline Query() {}
       inline Query(int a, int b, int c){
           idx = c;
           l = a, r = b, d = l / block_size;
       }

       inline bool operator < (const Query& other) const{
           if (d != other.d) return (d < other.d);
           return ((d & 1) ? (r < other.r) : (r > other.r));
       }
   } Q[MAX];

   /// Number of nodes, initial array, and block_size
   inline void init(int n, int *T, int sz){
       clr(freq);
       counter = 0, q = 0, block_size = max(1, sz);
       for (int i = 0; i < n; i++) ar[i] = T[i]; /// Change here if 1-
based array
   }

   inline void push(int a, int b, int idx){
       Q[q++] = Query(a, b, idx);
   }

   inline void insert(int idx){
       if (!freq[ar[idx]]) counter++;
       freq[ar[idx]]++;
   }

   inline void erase(int idx){
       freq[ar[idx]]--;
       if (!freq[ar[idx]]) counter--;
   }

   inline void run(){
       sort(Q, Q + q);
       int i, l, r, a = 0, b = 0; /// Change here if 1-based array

       for (counter = 0, i = 0; i < q; i++){
           l = Q[i].l, r = Q[i].r;
           while (a > l) insert(--a);
           while (a < l) erase(a++);
           while (b <= r) insert(b++);
```

```
            while (b > (r + 1)) erase(--b);
            out[Q[i].idx] = counter;
        }
    }

    inline void print(){
        for (int i = 0; i < q; i++) printf("%d\n", out[i]);
    }
}

int n, ar[MAX];

int main(){
    int i, j, k, q, a, b;

    scanf("%d", &n);
    for (i = 0; i < n; i++) scanf("%d", &ar[i]);
    mo::init(n, ar, 447);

    scanf("%d", &q);
    for (i = 0; i < q; i++){
        scanf("%d %d", &a, &b);
        mo::push(--a, --b, i);
    }

    mo::run();
    mo::print();
    return 0;
}
```

Mobius Function.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define LEN 666666
#define MAX 10000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int len = 0, prime[LEN];
char mu[MAX] = {0}, flag[MAX] = {0};

/// mu[1] = 1, mu[n] = 0 if n has a squared prime factor,
/// mu[n] = 1 if n is square-free with even number of prime factors
/// mu[n] = -1 if n is square-free with odd number of prime factors

void Mobius(){
    mu[1] = 1;
    int i, j, k;

    for (i = 2; i < MAX; i++){
        if (!flag[i]) mu[i] = -1, prime[len++] = i;
```

```
        for (j = 0; j < len && (k = i * prime[j]) < MAX; j++){

            flag[k] = true;
            if (!(i % prime[j])){
                mu[k] = 0;
                break;
            }
            else mu[k] -= mu[i];
        }
    }
}

void Mobius(){ /// Simplified NlogN version
    int i, j;

    mu[1] = 1;
    for (i = 1; i < MAX; i++){
        for (j = i + i; j < MAX; j += i){
            mu[j] -= mu[i];
        }
    }
}

void Mobius(){ /// Simplified version optimized
    int i, j;

    mu[1] = 1;
    for (i = 1; i < MAX; i++){
        if (mu[i]){
            for (j = i + i; j < MAX; j += i){
                mu[j] -= mu[i];
            }
        }
    }
}

int main(){
    Mobius();
}
```

Modular Inverse.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

int extended_gcd(int a, int b, int& x, int& y){
    /// Bezout's identity, ax + by = gcd(a,b)

    if (!b){
```

```
        y = 0, x = 1;
        return a;
    }

    int g = extended_gcd(b, a % b, y, x);
    y -= ((a / b) * x);
    return g;
}

int mod_inverse(int a, int m){
    /// inverse exists if and only if a and m are co-prime

    int x, y, inv;
    extended_gcd(a, m, x, y);
    inv = (x + m) % m;
    return inv;
}

int mod_inverse(int a, int mod){
    int x, y, u, v, q, r, m, n, b;

    u = y = 0, x = v = 1, b = mod;
    while (a){
        q = (b / a), r = (b % a);
        m = u - (q * x), n = v - (q * y);
        u = x, v = y;
        x = m, y = n;
        b = a, a = r;
    }

    if (u < 0) u += mod;
    return u;
}

/* Caution, crashes when m = 1, crashes sometimes when a = m
  Sometimes returns 0 if a and m are not co-primes
  Unstable but extremely elegant
*/

int mod_inverse(int a, int m){
    if ((a < 2)) return a;
    int res = (((1LL - (long long)m * mod_inverse(m % a, a)) / (a % m)) +
m) % m;
    return res;
}

int main(){
}
```

N Queen.c

```
#include <stdio.h>

int n, lim, counter;
```

```
void backtrack(int i, int c, int l, int r){
   if (!i){
       counter++;
       return;
   }

   int bitmask, x;
   --i, bitmask = lim & ~(l | r | c);

   while (bitmask){
       x = (-bitmask & bitmask);
       if (!x) return;
       bitmask ^= x;
       backtrack(i, c | x, (l | x) << 1, (r | x) >> 1);
   }
}

int main(){
   while (scanf("%d", &n)){
       counter = 0, lim = (1 << n) - 1;
       backtrack(n, 0, 0, 0);
       printf("%d solutions for a %d x %d chessboard\n", counter, n, n);
   }
   return 0;
}
```

Network Simplex OP.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define EPS 0
#define MAXV 4010
#define MAXC 4010

#define EQUAL 0
#define LESSEQ -1
#define GREATEQ 1
#define MINIMIZE -1
#define MAXIMIZE +1

#define FEASIBLE +1
#define INFEASIBLE -1
#define UNBOUNDED 666

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Network Simplex
/// Variables and constraints are 1 based
```

```
/// Solution is contained in val[] array after termination
/// Only works for special types of linear programs, for example as in
flow networks

namespace lp{
   long long ar[MAXC][MAXV], val[MAXV], rhs[MAXC];
   int n, m, flag, adj[MAXV], idx[MAXV], down[MAXV], link[MAXC];

   void init(int nvar, long long func[], int min_or_max){ /// func[] =
objective function
       m = 0, n = nvar, flag = min_or_max;
       for (int i = 1; i <= n; i++) idx[i] = 0;
       for (int i = 1; i <= n; i++) ar[0][i] = func[i] * flag;
   }

   /// var[] = co-efficients of the constraints (LHS), lim = limit in RHS
   inline void add_constraint(long long var[], long long lim, int flag){
       flag *= -1;
       if (flag == 0){
           rhs[++m] = lim;
           for (int i = 1; i <= n; i++) ar[m][i] = var[i];
           rhs[++m] = -lim;
           for (int i = 1; i <= n; i++) ar[m][i] = -var[i];
       }
       else{
           rhs[++m] = lim * flag;
           for (int i = 1; i <= n; i++) ar[m][i] = var[i] * flag;
       }
   }

   void pivot(int x, int y, long long& res){ /// pivoting and exchanging
a non-basic variable with a basic variable
           int i, j, len = 0;
       long long v = ar[x][y];

       swap(link[x], down[y]);
       rhs[x] /= v, ar[x][y] = 1;
           for (j = 1; j <= n; j++){
           ar[x][j] /= v;
           if (abs(ar[x][j]) > EPS) adj[len++] = j;
           }

           for (i = 1; i <= m; i++){
           if (abs(ar[i][y]) > EPS && i != x){
               rhs[i] -= ar[i][y] * rhs[x], v = ar[i][y], ar[i][y] = 0;
               for (j = 0; j < len; j++) ar[i][adj[j]] -= (v *
ar[x][adj[j]]);
           }
           }

           res += (ar[0][y] * rhs[x]), v = ar[0][y], ar[0][y] = 0;
           for (j = 0; j < len; j++) ar[0][adj[j]] -= (v *
ar[x][adj[j]]);
   }
```

```
    int solve(long long& res){ /// simplex core
        int i, j, x, y;
        long long u, v, mn, mx;
        for (i = 1; i <= n; i++) down[i] = i;
        for (i = 1; i <= m; i++) link[i] = i + n;

        while (1){ /// phase 1
            x = 0, y = 0, mn = -EPS;
            for (i = 1; i <= m; i++){
                if (rhs[i] < mn) mn = rhs[i], x = i;
            }
            if (x == 0) break;

            for (i = 1; i <= n; i++){
                if (ar[x][i] < -EPS){
                    y = i;
                    if (rand() & 1) break;
                }
            }
            if (y == 0) return INFEASIBLE;
            pivot(x, y, res);
        }

        while (1){ /// phase 2
            x = 0, y = 0, mx = EPS;
            for (i = 1; i <= n; i++){
                if (ar[0][i] > mx) mx = ar[0][i], y = i;
            }
            if (y == 0) break;

            for (i = 1; i <= m; i++){
                if (ar[i][y] > EPS){
                    u = rhs[i] / ar[i][y];
                    if (x == 0 || u < v) x = i, v = u;
                }
            }
            if (x == 0) return UNBOUNDED;
            pivot(x, y, res);
        }

        res *= flag;
        for (int i = 1; i <= m; i++){
            if(link[i] <= n) idx[link[i]] = i;
        }

        for (int i = 1; i <= n; i++) val[i] = rhs[idx[i]];
             return FEASIBLE;
      }
}

/// Reduces constraints in the form of a *(e.g. <, <=, >, >=) b, by
removing redundant constraints
/// Lim = number of random iterations
```

```
/// Returns false if contradictions exist, for example a < b and b < a

bool reduce(int n, int lim, vector <pair<int, int> >& edge){
   bitset <202> adj[n + 1]; /// Maximum value for n
   while (lim--){
       vector <pair<int, int> > temp = edge;
       random_shuffle(temp.begin(), temp.end());

       for (int i = 1; i <= n; i++){
           adj[i].reset();
           adj[i][i] = true;
       }

       edge.clear();
       int len = temp.size();
       for (int e = 0; e < len; e++){
           int i = temp[e].first, j = temp[e].second;
           if (!adj[i][j]){
               edge.push_back(temp[e]);
               if (adj[j][i]) return false;

               for (int k = 1; k <= n; k++){
                   if (adj[k][i]) adj[k] |= adj[j];
               }
           }
       }

       if (edge.size() == temp.size()) return true;
   }
   return true;
}

int main(){

}

Network Simplex.cpp
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAXV 1010
#define MAXC 10010

#define EQUAL 0
#define LESSEQ -1
#define GREATEQ 1
#define MINIMIZE -1
#define MAXIMIZE +1
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;
```

```
/// Network Simplex
/// Variables and constraints are 1 based
/// Solution is contained in val[] array after termination
/// Only works for special types of linear programs, for example as in
flow networks

namespace lp{
   bool has_solution;
   int n, m, flag, link[MAXC], down[MAXV], idx[MAXV];
   long long res, ar[MAXC][MAXV], val[MAXV], rhs[MAXC];

   void init(int nvar, long long func[], int min_or_max){ /// func[] =
objective function
       flag = min_or_max;
       has_solution = false;
       res = 0, m = 0, n = nvar;
       for (int i = 1; i <= n; i++) idx[i] = 0;
       for (int i = 1; i <= n; i++) ar[0][i] = func[i] * flag;
   }

   /// var[] = co-efficients of the constraints (LHS), lim = limit in RHS
   inline void add_constraint(long long var[], long long lim, int flag){
       flag *= -1;
       if (flag == 0){
           rhs[++m] = lim;
           for (int i = 1; i <= n; i++) ar[m][i] = var[i];
           rhs[++m] = -lim;
           for (int i = 1; i <= n; i++) ar[m][i] = -var[i];
       }
       else{
           rhs[++m] = lim * flag;
           for (int i = 1; i <= n; i++) ar[m][i] = var[i] * flag;
       }
   }

   void pivot(int x, int y){ /// pivoting and exchanging a non-basic
variable with a basic variable
            int i, j, t = 0;
       long long v = ar[x][y];

       rhs[x] /= v;
       ar[x][y] = 1;
       swap(link[x], down[y]);
            for (j = 1; j <= n; j++){
           ar[x][j] /= v;
           if (ar[x][j]) val[++t] = j;
            }

            for (i = 1; i <= m; i++){
           if (ar[i][y] && i != x){
               rhs[i] -= (ar[i][y] * rhs[x]);
               v = ar[i][y], ar[i][y] = 0;
```

```
                for (j = 1; j <= t; j++) ar[i][val[j]] -= (v *
ar[x][val[j]]);
            }
        }

        res += (ar[0][y] * rhs[x]), v = ar[0][y], ar[0][y] = 0;
        for (j = 1; j <= t; j++) ar[0][val[j]] -= (v *
ar[x][val[j]]);
    }

    long long solve(){ /// simplex core
        int i, j, x, y;
        long long u, v;
        for (i = 1; i <= n; i++) down[i] = i;
        for (i = 1; i <= m; i++) link[i] = i + n;

        while (1){
            for (x = 1; x <= m && rhs[x] >= 0; x++){}
            if (x > m) break;
            for (j = 1, y = 0; j <= n; j++){
                if (ar[x][j] < 0){
                    y = j;
                    if (rand() & 1) break; /// try removing rand()
                }
            }
            if (y == 0) return -666;
            pivot(x, y);
        }

        while (1){
            for (i = 1, y = 0; i <= n; i++){
                if (ar[0][i] > 0 && (y == 0 || ar[0][i] > ar[0][y])) y =
i;
            }
            if (y == 0) break;
            for (j = 1, x = 0; j <= m; j++){
                if (ar[j][y] > 0){
                    v = rhs[j] / ar[j][y];
                    if (x == 0 || v < u) x = j, u = v;
                }
            }
            assert(x != 0);
            pivot(x, y);
        }

        has_solution = true;
        for (int i = 1; i <= m; i++){
            if(link[i] <= n) idx[link[i]] = i;
        }
        for (int i = 1; i <= n; i++) val[i] = rhs[idx[i]];
            return res * flag;
        }
}
```

```
/// Reduces constraints in the form of a *(e.g. <, <=, >, >=) b, by
removing redundant constraints
/// Lim = number of random iterations
/// Returns false if contradictions exist, for example a < b and b < a

bool reduce(int n, int lim, vector <pair<int, int> >& edge){
   bitset <202> adj[n + 1]; /// Maximum value for n
   while (lim--){
        vector <pair<int, int> > temp = edge;
        random_shuffle(temp.begin(), temp.end());

        for (int i = 1; i <= n; i++){
            adj[i].reset();
            adj[i][i] = true;
        }

        edge.clear();
        int len = temp.size();
        for (int e = 0; e < len; e++){
            int i = temp[e].first, j = temp[e].second;
            if (!adj[i][j]){
                edge.push_back(temp[e]);
                if (adj[j][i]) return false;

                for (int k = 1; k <= n; k++){
                    if (adj[k][i]) adj[k] |= adj[j];
                }
            }
        }

        if (edge.size() == temp.size()) return true;
   }
   return true;
}

int main(){

}
```

Next Palindrome.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

char str[MAX];

void next_palindrome(char* str){
   int i, j, k, l, x, n = strlen(str), carry = 1;
   for (i = 0, j = n - 1; j >= 0; i++, j--){
```

```
        if (carry){
            if (str[j] == 57) carry = 1, str[j] = 48;
            else carry = 0, str[j]++;
        }

        if (i > j) str[i] = str[j];
        else{
            if (str[i] < str[j]) carry = 1;
            str[j] = str[i];
        }
    }

    if (carry){
        str[n - 1] = 49;
        for (i = n - 1; i >= 0; i--) str[i + 1] = str[i];
        str[0] = 49, str[n + 1] = 0;
    }
}

int main(){
    int T = 0, t, i, j, k;

    scanf("%d", &t);
    while (t--){
        scanf("%s", str);
        next_palindrome(str);
        printf("Case %d: %s\n", ++T, str);
    }
    return 0;
}
```

Next Small.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 250010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int ar[MAX], L[MAX], R[MAX], stack[MAX], time[MAX];

void next_small(int n, int* ar, int* L){
    int i, j, k, l, x, top = 0;

    for (i = 0; i < n; i++){
        x = ar[i];
        if (top && stack[top] >= x){
            while (top && stack[top] >= x) k = time[top--];
            L[i] = (i - k + 2);
            stack[++top] = x;
            time[top] = k;
        }
```

```
        else{
            L[i] = 1;
            stack[++top] = x;
            time[top] = i + 1;
        }
    }
}
/*** L[i] contains maximum length of the range from i to the left such
that the minimum of this range
    is not less than ar[i].
    Similarly, R[i] contains maximum length of the range from i to the
right such that the minimum
    of this range is not less than ar[i]

    For example, ar[] = 5 3 4 3 1 2 6
                 L[]  = 1 2 1 4 5 1 1
                 R[]  = 1 3 1 1 3 2 1
***/

void fill(int n, int* ar, int* L, int* R){
   int i, j, k;
   for (i = 0; i < n; i++) L[i] = ar[n - i - 1];

   next_small(n, L, R);
   next_small(n, ar, L);

   i = 0, j = n - 1;
   while (i < j){
       k = R[i], R[i] = R[j], R[j] = k;
       i++, j--;
   }
}

int main(){
   int n, i, j, k;

   scanf("%d", &n);
   for (i = 0; i < n; i++) scanf("%d", &ar[i]);

   fill(n, ar, L, R);
   for (i = 0; i < n; i++) printf("%d ", ar[i]);
   puts("");
   for (i = 0; i < n; i++) printf("%d ", R[i]);
   puts("");
   for (i = 0; i < n; i++) printf("%d ", L[i]);
   puts("");
   return 0;
}
```

Ordered Multiset.cpp

```
#include <bits/stdtr1c++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
```

```
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;
using namespace __gnu_pbds;

/*** Needs C++11 or C++14 ***/

/// Treap supporting duplicating values in set
/// Maximum value of treap * ADD must fit in long long

struct Treap{ /// hash = 96814
   int len;
   const int ADD = 1000010;
   const int MAXVAL = 1000000010;
   tr1::unordered_map <long long, int> mp; /// Change to int if only int
in treap
   tree<long long, null_type, less<long long>, rb_tree_tag,
tree_order_statistics_node_update> T;

   Treap(){
       len = 0;
       T.clear(), mp.clear();
   }

   inline void clear(){
       len = 0;
       T.clear(), mp.clear();
   }

   inline void insert(long long x){
       len++, x += MAXVAL;
       int c = mp[x]++;
       T.insert((x * ADD) + c);
   }

   inline void erase(long long x){
       x += MAXVAL;
       int c = mp[x];
       if (c){
           c--, mp[x]--, len--;
           T.erase((x * ADD) + c);
       }
   }

   /// 1-based index, returns the K'th element in the treap, -1 if none
exists
   inline long long kth(int k){
       if (k < 1 || k > len) return -1;
       auto it = T.find_by_order(--k);
       return ((*it) / ADD) - MAXVAL;
   }
```

```
   /// Count of value < x in treap
   inline int count(long long x){
       x += MAXVAL;
       int c = mp[--x];
       return (T.order_of_key((x * ADD) + c));
   }

   /// Number of elements in treap
   inline int size(){
       return len;
   }
};

int main(){
}
```

Ordered Set.cpp

```
#include <bits/stdtr1c++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;
using namespace __gnu_pbds;

/*** Needs C++11 or C++14 ***/

class Treap{
   public:

   tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> T;

   Treap(){
       T.clear();
   }

   inline void insert(int x){
       T.insert(x);
   }

   inline void erase(int x){
       T.erase(x);
   }

   /// 1-based index, returns the K'th element in the treap, -1 if none
exists
   inline int kth(int k){
       if (k < 1) return -1;
```

```
        auto it = T.find_by_order(--k);
        if (it == T.end()) return -1;
        return *it;
    }

    /// Count of value < x in treap
    inline int count(int x){
        return (T.order_of_key(x));
    }

    /// Number of elements in treap
    inline int size(){
        return T.size();
    }
};

int main(){
}
```

Output Compression.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int base = 0, mp[256];
char digit[256], str[256], temp[256];

void Generate(){
    int i;
    for (i = 32; i < 127; i++){
        if (i == 32 || i == 34 || i == 39 || i == 44 || i == 92) continue;

        digit[base] = i;
        mp[i] = base;
        base++;
    }
    digit[base] = 0;
}

void encode(char* str, int v){
    int i, j, k = 0, l = 0;
    do{
        temp[k++] = digit[v % base];
        v /= base;
    } while (v);

    for (i = k - 1; i >= 0; i--) str[l++] = temp[i];
    str[l] = 0;
}
```

```
int decode(char* str){
   int i, v = 0;
   for (i = 0; str[i]; i++) v = (v * base) + mp[str[i]];
   return v;
}

int main(){
   read();
   Generate();
   encode(str, 12345);
   printf("%d\n", decode(str));
   return 0;
}
```

Overflow.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MOD 1007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

long long mul(long long a, long long b){
   long long res = 0;

   while (b){
       if (b & 1LL) res = (res + a) % MOD;
       a = (a << 1LL) % MOD;
       b >>= 1LL;
   }

   return res;
}

long long mul(long long a, long long b){
  if ((MOD < 3037000500LL)) return ((a * b) % MOD);
  long double res = a;
  res *= b;
  long long c = (long long)(res / MOD);
  a *= b;
  a -= c * MOD;
  if (a >= MOD) a -= MOD;
  if (a < 0) a += MOD;
  return a;
}

long long binary_div(long long a, long long b){ /* (a + b) / 2 without
overflow */
   long long x = (a >> 1LL), y = (b >> 1LL);
   long long res = x + y;
   if ((a & 1) && (b & 1)) res++;
   return res;
```

```
}



const long long LIM = LLONG_MAX;

uint64_t mul(uint64_t a, uint64_t b){
   a %= MOD, b %= MOD;
   if (a > b) swap(a, b);
   if (!a) return 0;
   if (a < (LIM / b)) return ((a * b) % MOD);

   uint64_t res = 0;
   int x, k = min(30, __builtin_clzll(MOD) - 1);
   int bitmask = (1 << k) - 1;

   while (a > 0){
       x = a & bitmask;
       res = (res + (b * x)) % MOD;
       a >>= k;
       b = (b << k) % MOD;
   }
   return res;
}

/// Not sure, morris vesion
long long mul(long long a, long long b, long long MOD) {
   long long x = (long long)((double)a * b / MOD + 0.5);
   long long res = ((a * b) - (x * MOD)) % MOD;
   if (res < 0) res += MOD;
   return res;
}

int main(){
}
```

Pairsum.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/// SPOJ MAKESUM
/// Returns lexicographically smallest positive solution in vector res
for pairsum vector ar, false if none exists

bool Pairsum(vector <int> ar, vector <int>& res){
   res.clear();
   sort(ar.begin(), ar.end());
```

```
    int i, j, k, l, a, b, x, n = ar.size();

    if (n == 0){
        res.push_back(1); /// Since only positive solution is required
        return true;
    }
    if (n == 1){
        if (ar[0] < 2) return false; /// Since only positive solution is
required
        res.push_back(1), res.push_back(ar[0] - 1); /// Since only
positive solution is required
        return true;
    }

    res.resize(n, 0);
    a = ar[0], b = ar[1];
    vector <int> lex_smallest;

    for (i = 2, k = 2; i * (i - 1) < 2 * n && k < n; i++, k++){
        if ((a + b - ar[k]) & 1 ^ 1){
            j = 1, res[0] = (a + b - ar[k]) >> 1;
            multiset <int> S(ar.begin(), ar.end());

            while (S.size() && res[0] > 0){ /// Since only positive
solution is required
                res[j] = *S.begin() - res[0];
                for (l = 0; l < j && !S.empty(); l++){
                    auto pos = S.find(res[l] + res[j]);
                    if (pos == S.end()) goto skip;
                    S.erase(pos);
                }
                j++;
            }

            skip:
            if (j * (j - 1) == 2 * n){
                res.resize(j);
                sort(res.begin(), res.end());
                if (!lex_smallest.size() || res < lex_smallest)
lex_smallest = res;
            }
        }
    }
    res = lex_smallest;
    return (lex_smallest.size() > 0);
}

/// UVA Pairsumonious Numbers
/// Returns any valid solution in vector res for pairsum vector ar, false
if none exists

bool Pairsum(vector <int> ar, vector <int>& res){
    res.clear();
    sort(ar.begin(), ar.end());
```

```
    int i, j, k, l, a, b, x, n = ar.size();

    if (n == 0){
        res.push_back(1);
        return true;
    }
    if (n == 1){
        if (ar[0] < 2) return false;
        res.push_back(1), res.push_back(ar[0] - 1);
        return true;
    }

    res.resize(n, 0);
    a = ar[0], b = ar[1];
    for (i = 2, k = 2; i * (i - 1) < 2 * n && k < n; i++, k++){
        if ((a + b - ar[k]) & 1 ^ 1){
            j = 1, res[0] = (a + b - ar[k]) >> 1;
            multiset <int> S(ar.begin(), ar.end());

            while (S.size()){
                res[j] = *S.begin() - res[0];
                for (l = 0; l < j && !S.empty(); l++){
                    auto pos = S.find(res[l] + res[j]);
                    if (pos == S.end()) goto skip;
                    S.erase(pos);
                }
                j++;
            }

            skip:
            if (j * (j - 1) == 2 * n){
                res.resize(j);
                sort(res.begin(), res.end());
                return true;
            }
        }
    }
    return false;
}

int main(){
    vector <int> res;
    vector <int> ar = {216, 210, 204, 212, 220, 214, 222, 208, 216, 210};

    if (Pairsum(ar, res)){
        for (i = 0; i < res.size(); i++) printf("%d ", res[i]); /// output
= 101 103 107 109 113
        puts("");
    }
    return 0;
}
```

Palindrome Factorization.cpp
```
--------------------------------------------------
```

```
#include <bits/stdtr1c++.h>

#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Minimum palindromic factorization for all prefixes in O(N log N)

namespace pal{
   int pl[MAX][2], gpl[MAX][2];

   inline void set(int *ar, int x, int y, int z) {
       ar[0] = x, ar[1] = y, ar[2] = z;
   }

   inline void set(int ar[][2], int i, int v) {
       if (v > 0) ar[i][v & 1] = v;
   }

   inline void copy(int *A, int *B) {
       A[0] = B[0], A[1] = B[1], A[2] = B[2];
   }

   inline void update(int ar[][2], int i, int v) {
       if (v > 0 && (ar[i][v & 1] == -1 || ar[i][v & 1] > v)) ar[i][v &
1] = v;
   }

   /// Returns a vector v such that,
   /// v[i] is the minimum k so that the prefix string str[0:i] can be
partitioned into k disjoint palindromes
   inline vector <int> factorize(const char* str){
       int g[32][3], gp[32][3], gpp[32][3];
       int i, j, k, l, d, u, r, x, pg = 0, pgp = 0, pgpp = 0, n =
strlen(str);

       clr(g), clr(gp), clr(gpp);
       for (int i = 0; i < n; i++) gpl[i][0] = MAX, gpl[i][1] = MAX + 1;

       for (j = 0; j < n; j++){
           for (u = 0, pgp = 0; u < pg; u++){
               i = g[u][0];
               if ((i - 1) >= 0 && str[i - 1] == str[j]){
                   g[u][0]--;
                   copy(gp[pgp++], g[u]);
               }
           }

           pgpp = 0, r = -(j + 2);
           for (u = 0; u < pgp; u++){
```

```
        i = gp[u][0], d = gp[u][1], k = gp[u][2];
        if ((i - r) != d){
            set(gpp[pgpp++], i, i - r, 1);
            if (k > 1) set(gpp[pgpp++], i + d, d, k - 1);
        }
        else set(gpp[pgpp++], i, d, k);
        r = i + (k - 1) * d;
    }

    if (j - 1 >= 0 && str[j - 1] == str[j]){
        set(gpp[pgpp++], j - 1, j - 1 - r, 1);
        r = j - 1;
    }
    set(gpp[pgpp++], j, j - r, 1);

    int *ptr = gpp[0];
    for (u = 1, pg = 0; u < pgpp; u++){
        int *x = gpp[u];
        if (x[1] == ptr[1]) ptr[2] += x[2];
        else {
            copy(g[pg++], ptr);
            ptr = x;
        }
    }
    copy(g[pg++], ptr);

    pl[j + 1][(j & 1) ^ 1] = j + 1;
    pl[j + 1][j & 1] = MAX + (j & 1);
    for (u = 0; u < pg; u++){
        i = g[u][0], d = g[u][1], k = g[u][2];
        r = i + (k - 1) * d;

        update(pl, j + 1, pl[r][0] + 1);
        update(pl, j + 1, pl[r][1] + 1);
        if (k > 1) {
            update(pl, j + 1, gpl[i + 1 - d][0]);
            update(pl, j + 1, gpl[i + 1 - d][1]);
        }

        if (i + 1 >= d) {
            if (k > 1) {
                update(gpl, i + 1 - d, pl[r][0] + 1);
                update(gpl, i + 1 - d, pl[r][1] + 1);
            }
            else{
                set(gpl, i + 1 - d, pl[r][0] + 1);
                set(gpl, i + 1 - d, pl[r][1] + 1);
            }
        }
    }
}

vector <int> res(n, 0);
for (i = 0; i < n; i++) res[i] = min(pl[i + 1][0], pl[i + 1][1]);
```

```
        return res;
    }
}

int main(){

}
```

Palindrome Tree.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define LET 26
#define MAX 200010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

char str[MAX];

namespace ptree{ /// hash = 232659
    char S[MAX];
    int idx, cur, slen, lps, len[MAX], link[MAX], counter[MAX],
trie[MAX][LET];
    /// len[i] = length of the palindrome at node i, link[i] = suffix link
of node i
    /// link[i] = v, where v is the node representing the longest proper
suffix-palindrome of i
    /// trie[i][c] = node where the palindrome is c + palindrome at i + c,
e.g, i = "aba", c = 'd', trie[i][c] = "dabad"

    void init(){
        cur = 0, slen = 1, lps = 0, idx = 2;
        clr(S), clr(len), clr(link), clr(trie), clr(counter);

        S[0] = -1;
        link[0] = 1, len[0] = 0;
        link[1] = 0, len[1] = -1;
    }

    inline int nextlink(int cur){ /// Matching similar to fail function as
in KMP/Aho-corasick
        while (S[slen - len[cur] - 2] != S[slen - 1]) cur = link[cur];
        return cur;
    }

    /// Returns true if a new distinct palindromic substring appears after
adding current character
    inline bool insert(char ch){
        S[slen++] = ch, cur = nextlink(cur);
        int c = ch - 'a', flag = trie[cur][c]; /// Change here if any non-
lower case character can occur
```

```
        if (!flag){
            len[idx] = len[cur] + 2;
            link[idx] = trie[nextlink(link[cur])][c];
            trie[cur][c] = idx++;
        }

        cur = trie[cur][c];
        counter[cur]++; /// count of palindromic substring cur in the
string
        lps = max(lps, len[cur]); /// Update the longest palindromic
substring after adding this character
        return !flag;
    }

    /// IMPORTANT: do this in main to update count of all palindromic
nodes
    for (int i = idx; i >= 0; i--) counter[link[i]] += counter[i];
}

int main(){

}
```

Period.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

vector <int> period(char* str){
    int i, l = 0, r = 0, n = strlen(str);
    vector <int> Z(n + 1, 0);

    for (i = 1; i < n; i++){
        Z[i] = max(0, min(Z[i - l], r - i));
        while (str[i + Z[i]] && str[Z[i]] == str[i + Z[i]]) Z[i]++;
        if ((i + Z[i]) > r) l = i, r = i + Z[i];
    }
    Z[0] = n;

    vector <int> v;
    for (l = 1; l <= n; l++){
        if ((n % l) == 0){
            for (i = 0; i < n && Z[i] >= l; i += l){}
            if (i == n) v.push_back(l);
        }
    }
    return v;
}
```

```
int main(){
   vector <int> v = period("abaabaabaaba");
   for (int i = 0; i < v.size(); i++) dbg(v[i]);
   return 0;
}
```

Permutation Index.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 12
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define hash(ar)  dp[ar[0]][ar[1]][ar[2]][ar[3]][ar[4]][ar[5]]
#define lexic(ar) pos[ar[0]][ar[1]][ar[2]][ar[3]][ar[4]][ar[5]]

char ar[MAX];
int n, m, counter, factorial[MAX];
int last[1 << MAX], dp[MAX][MAX][MAX][MAX][MAX][MAX],
pos[MAX][MAX][MAX][MAX][MAX][MAX];

void backtrack(int i, int bitmask, int flag){
   if (i == m){
       if (flag & 1) hash(ar) = ++counter;
       if (flag & 2) lexic(ar) = last[bitmask]++;
       return;
   }

   int j;
   for (j = 0; j < n; j++){
       if (!(bitmask & (1 << j))){
           ar[i] = j;
           backtrack(i + 1, bitmask | (1 << j), flag);
       }
   }
}

void Generate(){
   int i, j;
   clr(ar), clr(last);
   counter = 0, m = n >> 1, factorial[0] = 1;
   for (i = 1; i < MAX; i++) factorial[i] = factorial[i - 1] * i;

   if (n & 1){
       backtrack(0, 0, 1);
       m++;
       backtrack(0, 0, 2);
       m--;
   }
   else backtrack(0, 0, 3);
}
```

```
int F(int P[MAX]){
   int i, a = 0, b = 0;
   char A[6] = {0}, B[6] = {0};
   for (i = 0; i < m; i++) A[a++] = P[i];
   for (i = m; i < n; i++) B[b++] = P[i];
   return ((hash(A) - 1) * factorial[b]) + lexic(B) + 1;
}

int main(){
   int i, j, P[MAX];

   while (scanf("%d", &n) != EOF){
       Generate();
       for (i = 0; i < n; i++) scanf("%d", &P[i]);
       for (i = 0; i < n; i++) P[i]--;
       int res = F(P);
       printf("%d\n", res);
   }
   return 0;
}
```

Permutation Rank.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Find's the rank of permutation
/// Rank and permutation elements are 1 base indexed

long long find_rank(vector <int> permutation){
   long long res = 1;
   int i, j, n = permutation.size();
   vector <bool> visited(n + 1, false);
   vector <long long> factorial(n + 1, 1);
   for (i = 1; i <= n; i++) factorial[i] = factorial[i - 1] * i;

   for (i = 1; i <= n; i++){
       for (j = 1; j <= n; j++){
           if (!visited[j]){
               if (permutation[i - 1] == j){
                   visited[j] = true;
                   break;
               }
               res += factorial[n - i];
           }
       }
   }
```

```
    return res;
}

/// Find's the k'th permutation of 1 to n
/// Rank and permutation elements are 1 base indexed

vector <int> find_rank(int n, long long k){
    int i, j;
    vector <int> res(n, 0);
    vector <bool> visited(n + 1, false);
    vector <long long> factorial(n + 1, 1);
    for (i = 1; i <= n; i++) factorial[i] = factorial[i - 1] * i;

    for (i = 1; i <= n; i++){
        for (j = 1; j <= n; j++){
            if (!visited[j]){
                if (factorial[n - i] >= k){
                    visited[j] = true;
                    res[i - 1] = j;
                    break;
                }
                k -= factorial[n - i];
            }
        }
    }
    return res;
}

int main(){
    return 0;
}
```

Phi OP.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 100000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int phi[MAX];
short P[MAX] = {0};

void Generate(){
    int i, j, k, d, x;

    P[0] = P[1] = 1;
    for (i = 4; i < MAX; i++, i++) P[i] = 2;

    for (i = 3; i * i < MAX;){
        d = i << 1;
        for (j = (i * i); j < MAX; j += d) P[j] = i;
```

```
        do{
            i++;
        } while (P[++i]);
    }

    phi[1] = 1;
    for (i = 2; i < MAX; i++){
        if (!P[i]) phi[i] = i - 1;
        else{
            k = i / P[i];
            if (!(k % P[i])) phi[i] = phi[k] * P[i];
            else phi[i] = phi[k] * (P[i] - 1);
        }
    }
}

int main(){
    Generate();
    return 0;
}
```

Phi.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 4000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int phi[MAX];

void Totient(){
    clr(phi);
    int i, j, x;

    for (i = 1; i < MAX; i++){
        phi[i] += i;
        for (j = (i << 1); j < MAX; j += i){
            phi[j] -= phi[i];
        }
    }
}

void Totient(){
    clr(phi);
    int i, j, x;

    for (i = 2; i < MAX; i++){
        if (!phi[i]){
            phi[i] = i - 1;
            for (j = (i << 1); j < MAX; j += i){
                x = phi[j];
```

```
                if (!x) x = j;
                x = (x / i) * (i - 1);
                phi[j] = x;
            }
        }
    }
}

int main(){
   Totient();
}
```

Pick_s Theorem.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

struct Point{
   long long x, y;
   Point(){}
   Point(long long x, long long y) : x(x), y(y){}
};

/// twice the area of polygon
long long double_area(Point poly[], int n){
   long long res = 0;
   for (int i = 0, j = n - 1; i < n; j = i++){
       res += ((poly[j].x + poly[i].x) * (poly[j].y - poly[i].y));
   }
   return abs(res);
}

/// number of lattice points strictly on polygon border
long long on_border(Point poly[], int n){
   long long res = 0;
   for (int i = 0, j = n - 1; i < n; j = i++){
       res += __gcd(abs(poly[i].x - poly[j].x), abs(poly[i].y -
poly[j].y));
   }
   return res;
}

/// number of lattice points strictly inside polygon
long long interior(Point poly[], int n){
   long long res = 2 + double_area(poly, n) - on_border(poly, n);
   return res / 2;
}
```

```
int main(){
   Point ar[10];
   ar[0] = Point(0, 0);
   ar[1] = Point(3, 0);
   ar[2] = Point(3, 3);
   ar[3] = Point(0, 3);

   dbg(on_border(ar, 4)); /// 12
   dbg(interior(ar, 4)); /// 4
   return 0;
}
```

Point In Polygon.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

struct Point{
   long long x, y;

   Point(){
   }

   Point(long long xi, long long yi){
       x = xi, y = yi;
   }
};

struct Segment{
   struct Point P1, P2;

   Segment(){
   }

   Segment(struct Point P1i, struct Point P2i){
       P1 = P1i, P2 = P2i;
   }
};

/// Returns 0 if ABC is collinear, positive if ABC is a left turn,
negative if ABC is a right turn
long long ccw(struct Point A, struct Point B, struct Point C){
   return ((B.x - A.x) * (C.y - A.y)) - ((C.x - A.x) * (B.y - A.y));
}

/// Returns true if Point P lies on the Segment (both end-points
inclusive)
```

```
bool PointOnSeg(struct Segment S, struct Point P){
   long long x = P.x, y = P.y, x1 = S.P1.x, y1 = S.P1.y, x2 = S.P2.x, y2
= S.P2.y;
   long long a = x - x1, b = y - y1, c = x2 - x1, d = y2 - y1, dot = (a *
c) + (b * d), len = (c * c) + (d * d);

   if (x1 == x2 && y1 == y2) return (x1 == x && y1 == y);
   if (dot < 0 || dot > len) return false;
   return ((((x1 * len) + (dot * c)) == (x * len)) && (((y1 * len) + (dot
* d)) == (y * len)));
}

struct Polygon{
   #define CLOCKWISE 11230926
   #define ANTICLOCK 37281927

   int n; /// n should be greater than 1
   struct Point ar[MAX]; /// Points in polygon in clockwise order

   Polygon(){
   }

   /// Points in T are either in anti-clockwise or clockwise order
   Polygon(int ni, struct Point* T, int flag){
       n = ni;
       for (int i = 0; i < n; i++){
           if (flag == CLOCKWISE) ar[i] = T[i];
           else ar[i] = T[n - i - 1];
       }
   }

   /// strictly should be true if P needs to be strictly contained in the
polygon
   bool contains(struct Point P, bool strictly){
       int mid, low = 1, high = n - 1, idx = 1;

       while (low <= high){
           mid = (low + high) >> 1;
           if (ccw(ar[0], P, ar[mid]) > 0) low = mid + 1;
           else idx = mid, high = mid - 1;
       }

       if (!strictly && PointOnSeg(Segment(ar[0], ar[n - 1]), P)) return
true;
       if (!strictly && PointOnSeg(Segment(ar[idx], ar[idx - 1]), P))
return true;
       if (idx == 1 || ccw(ar[0], P, ar[n - 1]) == 0) return false;
       return (ccw(ar[idx], P, ar[idx - 1]) < 0);
   }
};

int main(){
}
```

Point Rotations.cpp

```cpp
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

struct Point{
   double x, y;

   Point(){
   }

   Point(double xi, double yi){
       x = xi, y = yi;
   }
};

double dis(struct Point A, struct Point B){
   double x = A.x - B.x;
   double y = A.y - B.y;
   return sqrt((x * x) + (y * y));
}

/// Rotate P anti-clockwise around the centre point by theta (theta in
degrees)
struct Point rotate(struct Point centre, struct Point P, double theta){
   P.x -= centre.x, P.y -= centre.y;
   theta = (theta * 2.0 * acos(0.0)) / 180.0;

   double s = sin(theta), c = cos(theta);
   double x = (P.x * c) - (P.y * s);
   double y = (P.x * s) + (P.y * c);
   return Point(x + centre.x, y + centre.y);
}

/// Clockwise rotation from vector A to vector B
double thetaX(struct Point A, struct Point B){
   double dot = (B.x * A.x) + (B.y * A.y);
   double det = (B.x * A.y) - (B.y * A.x);
   double theta = (atan2(det, dot) * 180.0) / (2.0 * acos(0.0));
   if (theta < 0) theta += 360.0;
   return theta;
}

int main(){

}
```

Pollard Rho OP 2.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 100000
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;
using namespace tr1;

const long long LIM = LLONG_MAX;

int p, P[MAX];
bitset <MAX> prime;
unordered_map <long long, long long> mp;

void Sieve(){
   prime.reset();
   int i, j, d;
   const int sqr = 325;

   prime[2] = true;
   for (i = 3; i < MAX; i++, i++) prime[i] = true;

   for (i = 3; i < sqr;){
       d = i << 1;
       for (j = (i * i); j < MAX; j += d) prime[j] = false;

       i++, i++;
       while (!prime[i]) i++, i++;
   }

   p = 0;
   for (i = 0; i < MAX; i++){
       if (prime[i]) P[p++] = i;
   }
}

uint64_t mul(uint64_t a, uint64_t b, uint64_t m){
   if (a > b) swap(a, b);
   if (!a) return 0;
   a %= m, b %= m;
   if (a < (LIM / b)) return ((a * b) % m);

   uint64_t res = 0;
   int x, k = min(30, __builtin_clzll(m));
   int bitmask = (1 << k) - 1;

   while (a > 0){
       x = a & bitmask;
       res = (res + (b * x)) % m;
       a >>= k;
       b = (b << k) % m;
```

```
    }
    return res;
}

long long expo(long long x, long long n, long long m){
    long long res = 1;

    while (n){
        if (n & 1) res = mul(res, x, m);
        x = mul(x, x, m);
        n >>= 1;
    }

    return (res % m);
}

bool miller_rabin(long long p, int lim){
    long long a, s, m, x, y;
    s = p - 1, y = p - 1;
    while (!(s & 1)) s >>= 1;

    while (lim--){
        x = s;
        a = (rand() % y) + 1;
        m = expo(a, x, p);

        while ((x != y) && (m != 1) && (m != y)){
            m = mul(m, m, p);
            x <<= 1;
        }
        if ((m != y) && !(x & 1)) return false;
    }

    return true;
}

long long gcd(long long u, long long v){
    if (!u || !v) return (u | v);

    int shift;
    for (shift = 0; !((u | v) & 1); shift++){
        u >>= 1;
        v >>= 1;
    }

    while (!(u & 1)) u >>= 1;
    do{
        while (!(v & 1)) v >>= 1;
        if (u < v) v -= u;
        else{
            long long d = u - v;
            u = v;
            v = d;
        }
```

```
        v >>= 1;
    }
    while (v);

    return (u << shift);
}

long long brent_pollard_rho(long long n){
    if (miller_rabin(n, 10)) return n;

    const long long m = 1000;
    long long i, k, a, x, y, ys, r, q, g;
    g = mp[n];
    if (g) return g;

    do{
        a = rand() % n;
    }
    while (a == 0 || a == (n - 2));

    r = 1, q = 1;
    y = rand() % n;

    do{
        x = y;
        for (i = 0; i < r; i++){
            y = mul(y, y, n);
            y += a;
            if (y < a) y += (ULLONG_MAX - n) + 1;
            y %= n;
        }

        k = 0;
        do{
            for (i = 0; (i < m) && (i < (r - k)); i++){
                ys = y;
                y = mul(y, y, n);
                y += a;
                if (y < a) y += (ULLONG_MAX - n) + 1;
                y %= n;
                q = mul(q, abs(x - y), n);
            }

            g = gcd(q, n);
            k += m;

        }
        while ((k < r) && (g == 1));

        r <<= 1;
    }
    while (g == 1);

    if (g == n){
```

```
        do{
            ys = mul(ys, ys, n);
            ys += a;
            if (ys < a) ys += (ULLONG_MAX - n) + 1;
            ys %= n;
            g = gcd(abs(x - ys), n);
        }
        while (g == 1);
    }

    return (mp[n] = g);
}

void factorize(long long n, vector <long long> &v){
    srand(time(0));

    int i, d, len;
    long long m, k, x;
    vector <long long> factors;

    for (i = 0; i < p; i++){
        d = P[i];
        while ((n % d) == 0){
            n /= d;
            v.push_back(d);
        }
    }
    if (n == 1) return;

    x = brent_pollard_rho(n);
    factors.push_back(x);
    factors.push_back(n / x);

    do{
        len = factors.size();
        m = factors[len - 1];
        factors.pop_back(), len--;

        if (m > 1){
            if (miller_rabin(m, 10)){
                v.push_back(m);
                for (i = 0; i < len; i++){
                    k = factors[i];
                    while ((k % m) == 0){
                        k /= m;
                        v.push_back(m);
                    }
                    factors[i] = k;
                }
            }
            else{
                x = brent_pollard_rho(m);
                factors.push_back(x);
                factors.push_back(m / x);
```

```
            }
        }
    }
    while (factors.size());
}

void Generate(){
    Sieve();
    mp.clear();
}

int main(){
    Generate();

    int i, t;
    long long n, x;

    scanf("%d", &t);
    while (t--){
        scanf("%lld", &n);

        vector <long long> v;
        factorize(n, v);
        sort(v.begin(), v.end());

        v.push_back(-1);
        int len = v.size();
        int c = 0, counter = 0;

        printf("%lld = ", n);
        for (i = 0; (i + 1) < len; i++){
            if (v[i] == v[i + 1]) counter++;
            else{
                if (c) printf(" * ");
                if (counter) printf("%lld^%d", v[i], ++counter);
                else printf("%lld", v[i]);
                c++, counter = 0;
            }
        }
        puts("");
    }
    return 0;
}
```

Pollard Rho OP.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 100000
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
```

```
using namespace std;

const long long LIM = LLONG_MAX;

int p, P[MAX];
bitset <MAX> prime;
unordered_map <long long, long long> mp;

void Sieve(){
   prime.reset();
   int i, j, d;
   const int sqr = 325;

   prime[2] = true;
   for (i = 3; i < MAX; i++, i++) prime[i] = true;

   for (i = 3; i < sqr;){
       d = i << 1;
       for (j = (i * i); j < MAX; j += d) prime[j] = false;

       i++, i++;
       while (!prime[i]) i++, i++;
   }

   p = 0;
   for (i = 0; i < MAX; i++){
       if (prime[i]) P[p++] = i;
   }
}

long long mul(long long a, long long b, long long m){
   long long x, res;

   if (a < b) swap(a, b);
   if (!b) return 0;
   if (a < (LIM / b)) return ((a * b) % m);

   res = 0, x = (a % m);
   while (b){
       if (b & 1){
           res = res + x;
           if (res >= m) res -= m;
       }

       b >>= 1;
       x <<= 1;
       if (x >= m) x -= m;
   }

   return res;
}

long long expo(long long x, long long n, long long m){
   long long res = 1;
```

```
    while (n){
        if (n & 1) res = mul(res, x, m);
        x = mul(x, x, m);
        n >>= 1;
    }

    return (res % m);
}

bool miller_rabin(long long p, int lim){
    long long a, s, m, x, y;
    s = p - 1, y = p - 1;
    while (!(s & 1)) s >>= 1;

    while (lim--){
        x = s;
        a = (rand() % y) + 1;
        m = expo(a, x, p);

        while ((x != y) && (m != 1) && (m != y)){
            m = mul(m, m, p);
            x <<= 1;
        }
        if ((m != y) && !(x & 1)) return false;
    }

    return true;
}

long long gcd(long long u, long long v){
    if (!u || !v) return (u | v);

    int shift;
    for (shift = 0; !((u | v) & 1); shift++){
        u >>= 1;
        v >>= 1;
    }

    while (!(u & 1)) u >>= 1;
    do{
        while (!(v & 1)) v >>= 1;
        if (u < v) v -= u;
        else{
            long long d = u - v;
            u = v;
            v = d;
        }
        v >>= 1;
    }
    while (v);

    return (u << shift);
}
```

```
long long brent_pollard_rho(long long n){
   if (miller_rabin(n, 10)) return n;

   const long long m = 1000;
   long long i, k, a, x, y, ys, r, q, g;
   g = mp[n];
   if (g) return g;

   do{
       a = rand() % n;
   }
   while (a == 0 || a == (n - 2));

   r = 1, q = 1;
   y = rand() % n;

   do{
       x = y;
       for (i = 0; i < r; i++){
           y = mul(y, y, n);
           y += a;
           if (y < a) y += (ULLONG_MAX - n) + 1;
           y %= n;
       }

       k = 0;
       do{
           for (i = 0; (i < m) && (i < (r - k)); i++){
               ys = y;
               y = mul(y, y, n);
               y += a;
               if (y < a) y += (ULLONG_MAX - n) + 1;
               y %= n;
               q = mul(q, abs(x - y), n);
           }

           g = gcd(q, n);
           k += m;

       }
       while ((k < r) && (g == 1));

       r <<= 1;
   }
   while (g == 1);

   if (g == n){
       do{
           ys = mul(ys, ys, n);
           ys += a;
           if (ys < a) ys += (ULLONG_MAX - n) + 1;
           ys %= n;
           g = gcd(abs(x - ys), n);
```

```
        }
        while (g == 1);
    }

    return (mp[n] = g);
}

void factorize(long long n, vector <long long> &v){
    srand(time(0));

    int i, d, len;
    long long m, k, x;
    vector <long long> factors;

    for (i = 0; i < p; i++){
        d = P[i];
        while ((n % d) == 0){
            n /= d;
            v.push_back(d);
        }
    }
    if (n == 1) return;

    x = brent_pollard_rho(n);
    factors.push_back(x);
    factors.push_back(n / x);

    do{
        len = factors.size();
        m = factors[len - 1];
        factors.pop_back(), len--;

        if (m > 1){
            if (miller_rabin(m, 10)){
                v.push_back(m);
                for (i = 0; i < len; i++){
                    k = factors[i];
                    while ((k % m) == 0){
                        k /= m;
                        v.push_back(m);
                    }
                    factors[i] = k;
                }
            }
            else{
                x = brent_pollard_rho(m);
                factors.push_back(x);
                factors.push_back(m / x);
            }
        }
    }
    while (factors.size());
}
```

```
void Generate(){
   Sieve();
   mp.clear();
}

int main(){
   Generate();

   int i, t;
   long long n, x;

   scanf("%d", &t);
   while (t--){
       scanf("%lld", &n);

       vector <long long> v;
       factorize(n, v);
       sort(v.begin(), v.end());

       v.push_back(-1);
       int len = v.size();
       int c = 0, counter = 0;

       printf("%lld = ", n);
       for (i = 0; (i + 1) < len; i++){
           if (v[i] == v[i + 1]) counter++;
           else{
               if (c) printf(" * ");
               if (counter) printf("%lld^%d", v[i], ++counter);
               else printf("%lld", v[i]);
               c++, counter = 0;
           }
       }
       puts("");
   }

   return 0;
}
```

Pollard Rho.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

const long long LIM = LLONG_MAX;

long long mul(long long a, long long b, long long m){
   long long x, res;
```

```
    if (a < b) swap(a, b);
    if (!b) return 0;
    if (a < (LIM / b)) return ((a * b) % m);

    res = 0, x = (a % m);
    while (b){
        if (b & 1){
            res = res + x;
            if (res >= m) res -= m;
        }
        b >>= 1;
        x <<= 1;
        if (x >= m) x -= m;
    }

    return res;
}

long long expo(long long x, long long n, long long m){
    long long res = 1;

    while (n){
        if (n & 1) res = mul(res, x, m);
        x = mul(x, x, m);
        n >>= 1;
    }

    return (res % m);
}

const int small_primes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
41, 43, 47, 51, 53, 59, 61, 67, 71};

bool miller_rabin(long long p, int lim){
    long long a, s, m, x, y;
    s = p - 1, y = p - 1;
    while (!(s & 1)) s >>= 1;

    while (lim--){
        x = s;
        a = (rand() % y) + 1;
        m = expo(a, x, p);

        while ((x != y) && (m != 1) && (m != y)){
            m = mul(m, m, p);
            x <<= 1;
        }
        if ((m != y) && !(x & 1)) return false;
    }

    return true;
}

void brent_pollard_rho(uint64_t n, vector <uint64_t> &v){
```

```
    if (miller_rabin(n, 10)){
        v.push_back(n);
        return;
    }

    uint64_t a, g, x, y;
    y = 1;
    a = rand() % n;
    x = rand() % n;

    for (int i = 0; ((i * i) >> 1) < n; i++){
        x = mul(x, x, n);
        x += a;
        if (x < a) x += (ULLONG_MAX - n) + 1;
        x %= n;

        g = __gcd(n, y - x);
        if ((g != 1) && (g != n)){
            n /= g;
            brent_pollard_rho(g, v);
            if (n != g) brent_pollard_rho(n, v);
            else if (miller_rabin(n, 10)) v.push_back(n);
            return;
        }

        if (!(i & (i - 1))) y = x;
    }
    brent_pollard_rho(n, v);
}

void factorize(uint64_t n, vector <uint64_t> &v){
    srand(time(0));
    int i, j, x;

    for (i = 0; i < 21; i++){
        x = small_primes[i];
        while ((n % x) == 0){
            n /= x;
            v.push_back(x);
        }
    }

    if (n > 1) brent_pollard_rho(n, v);
    sort(v.begin(), v.end());
}

int main(){
    int i, t;
    uint64_t n, x;

    cin >> t;
    while (t--){
        cin >> n;
        vector <uint64_t> v;
```

```
        factorize(n, v);
        sort(v.begin(), v.end());

        v.push_back(-1);
        int len = v.size();
        int c = 0, counter = 0;

        printf("%llu = ", n);
        for (i = 0; (i + 1) < len; i++){
            if (v[i] == v[i + 1]) counter++;
            else{
                if (c) printf(" * ");
                if (counter) printf("%llu^%d", v[i], ++counter);
                else printf("%llu", v[i]);
                c++, counter = 0;
            }
        }
        puts("");
    }

    return 0;
}
```

Prime Counting Functions (Prime Sums).cpp

```
#include <bits/stdtr1c++.h>

#define MAXN 100
#define MAXM 100010
#define MAXP 666666
#define MAX 10000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define chkbit(ar, i) (((ar[(i) >> 6]) & (1 << (((i) >> 1) & 31))))
#define setbit(ar, i) (((ar[(i) >> 6]) |= (1 << (((i) >> 1) & 31))))
#define isprime(x) (( (x) && ((x)&1) && (!chkbit(ar, (x)))) || ((x) ==
2))

using namespace std;

namespace pcf{
    long long sum[MAX], dp[MAXN][MAXM];
    unsigned int ar[(MAX >> 6) + 5] = {0};
    int len = 0, primes[MAXP], counter[MAX];

    void Sieve(){
        setbit(ar, 0), setbit(ar, 1);
        for (int i = 3; (i * i) < MAX; i++, i++){
            if (!chkbit(ar, i)){
                int k = i << 1;
                for (int j = (i * i); j < MAX; j += k) setbit(ar, j);
            }
        }
```

```
        for (int i = 1; i < MAX; i++){
            sum[i] = sum[i - 1], counter[i] = counter[i - 1];
            if (isprime(i)) primes[len++] = i, sum[i] += i, counter[i]++;
        }
    }

    void init(){
        Sieve();
        for (int n = 0; n < MAXN; n++){
            for (int m = 0; m < MAXM; m++){
                if (!n) dp[n][m] = ((long long)m * (m + 1)) >> 1;
                else dp[n][m] = dp[n - 1][m] - (dp[n - 1][m / primes[n -
1]] * primes[n - 1]);
            }
        }
    }

    long long phi(long long m, int n){
        if (!n) return (m * (m + 1)) >> 1;
        if (primes[n - 1] >= m) return 1;
        if (m < MAXM && n < MAXN) return dp[n][m];
        if ((long long)primes[n - 1] * primes[n - 1] >= m && m < MAX)
return sum[m] - sum[primes[n - 1]] + 1;
        return phi(m, n - 1) - (phi(m / primes[n - 1], n - 1) * primes[n -
1]);
    }

    long long Legendre(long long m){ /// Sum of all primes not greater
than m
        if (m < MAX) return sum[m];
        int lim = sqrt(0.9 + m);
        return phi(m, counter[lim]) + sum[lim] - 1;
    }

    long long Lehmer(long long m){ /// Sum of all primes not greater than
m
        if (m < MAX) return sum[m];

        long long w, res = 0;
        int i, a, s, c, x, y;
        s = sqrt(0.9 + m), y = c = cbrt(0.9 + m);
        a = counter[y], res = phi(m, a) + sum[y] - 1;

        for (i = a; primes[i] <= s; i++){
            w = Lehmer(m / primes[i]) - Lehmer(primes[i] - 1);
            res -= (w * primes[i]);
        }
        return res;
    }
}

int main(){
    pcf::init();
```

```
    clock_t start = clock();
    dbg(pcf::Lehmer(1e9)); /// 24739512092254535
    dbg(pcf::Lehmer(1e12));
    printf("%0.3f\n", (clock() - start) / (1.0 * CLOCKS_PER_SEC));
    return 0;
}
```

Prime Counting Functions LL (Simple).cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAXN 100
#define MAXM 100010
#define MAXP 666666
#define MAX 10000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define chkbit(ar, i) (((ar[(i) >> 6]) & (1 << (((i) >> 1) & 31))))
#define setbit(ar, i) (((ar[(i) >> 6]) |= (1 << (((i) >> 1) & 31))))
#define isprime(x) (( (x) && ((x)&1) && (!chkbit(ar, (x)))) || ((x) ==
2))

using namespace std;

namespace pcf{
    long long dp[MAXN][MAXM];
    unsigned int ar[(MAX >> 6) + 5] = {0};
    int len = 0, primes[MAXP], counter[MAX];

    void Sieve(){
        setbit(ar, 0), setbit(ar, 1);
        for (int i = 3; (i * i) < MAX; i++, i++){
            if (!chkbit(ar, i)){
                int k = i << 1;
                for (int j = (i * i); j < MAX; j += k) setbit(ar, j);
            }
        }

        for (int i = 1; i < MAX; i++){
            counter[i] = counter[i - 1];
            if (isprime(i)) primes[len++] = i, counter[i]++;
        }
    }

    void init(){
        Sieve();
        for (int n = 0; n < MAXN; n++){
            for (int m = 0; m < MAXM; m++){
                if (!n) dp[n][m] = m;
                else dp[n][m] = dp[n - 1][m] - dp[n - 1][m / primes[n -
1]];
            }
        }
```

```
    }

    /// Count of numbers not greater than m and not divisible by any of
the first n primes
    long long phi(long long m, int n){
        if (n == 0) return m;
        if (primes[n - 1] >= m) return 1;
        if (m < MAXM && n < MAXN) return dp[n][m];
        if ((long long)primes[n - 1] * primes[n - 1] >= m && m < MAX)
return counter[m] - n + 1;

        if ((long long)primes[n - 1] * primes[n - 1] * primes[n - 1] >= m
&& m < MAX){
            int lim = counter[(int)sqrt(m + 0.95)];
            long long res = counter[m] - (((lim + n - 2) * (lim - n + 1 ))
>> 1);
            for (int i = n; i < lim; i++) res += counter[m / primes[i]];
            return res;
        }

        return phi(m, n - 1) - phi(m / primes[n - 1], n - 1);
    }

    long long Legendre(long long m){ /// Returns the number of primes not
greater than m
        if (m < MAX) return counter[m];
        int lim = sqrt(0.9 + m);
        return phi(m, counter[lim]) + counter[lim] - 1;
    }

    long long Lehmer(long long m){ /// Returns the number of primes not
greater than m
        if (m < MAX) return counter[m];

        long long w, res = 0;
        int i, a, s, c, x, y;
        s = sqrt(0.9 + m), y = c = cbrt(0.9 + m);
        a = counter[y], res = phi(m, a) + a - 1;
        for (i = a; primes[i] <= s; i++) res = res - Lehmer(m / primes[i])
+ Lehmer(primes[i]) - 1;
        return res;
    }
}

int main(){
    pcf::init();
    clock_t start = clock();
    dbg(pcf::Lehmer(1e12)); /// 37607912018
    printf("%0.3f\n", (clock() - start) / (1.0 * CLOCKS_PER_SEC)); ///
0.422
    start = clock();
    dbg(pcf::Legendre(1e12)); /// 37607912018
    printf("%0.3f\n", (clock() - start) / (1.0 * CLOCKS_PER_SEC)); ///
1.079
```

```
    return 0;
}
```

Prime Counting Functions LL.cpp

```
---------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAXN 100
#define MAXM 100010
#define MAXP 666666
#define MAX 10000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define chkbit(ar, i) (((ar[(i) >> 6]) & (1 << (((i) >> 1) & 31))))
#define setbit(ar, i) (((ar[(i) >> 6]) |= (1 << (((i) >> 1) & 31))))
#define isprime(x) (( (x) && ((x)&1) && (!chkbit(ar, (x)))) || ((x) ==
2))

using namespace std;

namespace pcf{ /// Prime counting function
    long long dp[MAXN][MAXM];
    unsigned int ar[(MAX >> 6) + 5] = {0};
    int len = 0, primes[MAXP], counter[MAX];

    void Sieve(){
        setbit(ar, 0), setbit(ar, 1);
        for (int i = 3; (i * i) < MAX; i++, i++){
            if (!chkbit(ar, i)){
                int k = i << 1;
                for (int j = (i * i); j < MAX; j += k) setbit(ar, j);
            }
        }

        for (int i = 1; i < MAX; i++){
            counter[i] = counter[i - 1];
            if (isprime(i)) primes[len++] = i, counter[i]++;
        }
    }

    void init(){
        Sieve();
        for (int n = 0; n < MAXN; n++){
            for (int m = 0; m < MAXM; m++){
                if (!n) dp[n][m] = m;
                else dp[n][m] = dp[n - 1][m] - dp[n - 1][m / primes[n -
1]];
            }
        }
    }

    long long phi(long long m, int n){ /// Returns the number of primes
not greater than m
```

```
        if (n == 0) return m;
        if (primes[n - 1] >= m) return 1;
        if (m < MAXM && n < MAXN) return dp[n][m];
        return phi(m, n - 1) - phi(m / primes[n - 1], n - 1);
    }

    long long Legendre(long long m){
        if (m < MAX) return counter[m];
        int lim = sqrt(0.9 + m);
        return phi(m, counter[lim]) + counter[lim] - 1;
    }

    long long Lehmer(long long m){ /// Returns the number of primes not
greater than m
        if (m < MAX) return counter[m];

        int i, j, a, b, c, lim;
        b = sqrt(0.9 + m), c = Lehmer(cbrt(0.9 + m)), a = Lehmer(sqrt(0.9
+ b)), b = Lehmer(b);

        long long res = phi(m, a) + (( (long long)(b + a - 2) * (b - a +
1)) >> 1);
        for (i = a; i < b; i++){
            long long w = m / primes[i];
            lim = Lehmer(sqrt(0.9 + w)), res -= Lehmer(w);
            for (j = i; j < lim && i <= c; j++) res = res + j - Lehmer(w /
primes[j]);
        }

        return res;
    }
}

int main(){
    pcf::init();
    dbg(pcf::Lehmer(1e11)); /// 4118054813
    return 0;
}
```

Prime Counting Functions.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define chkbit(ar, i) (((ar[(i) >> 6]) & (1 << (((i) >> 1) & 31))))
#define setbit(ar, i) (((ar[(i) >> 6]) |= (1 << (((i) >> 1) & 31))))
#define isprime(x) (( (x) && ((x)&1) && (!chkbit(ar, (x)))) || ((x) ==
2))

using namespace std;

namespace pcf{
```

```
///   Prime-Counting Function
///   initialize once by calling init()
///   Legendre(m) and Lehmer(m) returns the number of primes less than
or equal to m

#define MAXN 1000010
#define MAX_PRIMES 1000010
#define PHI_M 10010
#define PHI_N 48

unsigned int ar[(MAXN >> 6) + 5] = {0};
int len = 0, primes[MAX_PRIMES], counter[MAXN], phi_dp[PHI_N][PHI_M];

void Sieve(int n, unsigned int* ar, int* primes, int& len){
    /// ar must be all set to 0

    n++;
    setbit(ar, 0), setbit(ar, 1);
    int i, j, k, lim = sqrt(n) + 1;

    for (i = 3; i < lim; i++, i++){
        if (!chkbit(ar, i)){
            k = i << 1;
            for (j = (i * i); j < n; j += k) setbit(ar, j);
        }
    }

    for (i = 1; i < n; i++){
        if (isprime(i)) primes[len++] = i;
        counter[i] = len;
    }
}

void Sieve(int n){
    Sieve(n, ar, primes, len);
}

void init(){ /// Call just once
    Sieve(MAXN - 1);

    int m, n, res;
    for (n = 0; n < PHI_N; n++){
        for (m = 0; m < PHI_M; m++){
            if (!n) res = m;
            else res = phi_dp[n - 1][m] - phi_dp[n - 1][m / primes[n -
1]];
            phi_dp[n][m] = res;
        }
    }
}

int phi(int m, int n){ /// long long
    if (m < PHI_M && n < PHI_N) return phi_dp[n][m];
```

```
        if (n == 1) return ((++m) >> 1);
        if (primes[n - 1] >= m) return 1;
        return phi(m, n - 1) - phi(m / primes[n - 1], n - 1);
    }


    int Legendre(int m){ /// long long
        if (m < MAXN) return counter[m];

        int lim = sqrt(m) + 1;
        int n = upper_bound(primes, primes + len, lim) - primes;
        return phi(m, n) + (n - 1);
    }


    int Lehmer(int m){ /// Very fast, 100 function calls for any integer
per second
        if (m < MAXN) return counter[m];

        int i, j, a, b, c, w, lim, res;
        b = sqrt(m), c = Lehmer(cbrt(m)), a = Lehmer(sqrt(b)), b =
Lehmer(b);
        res = phi(m, a) + (((b + a - 2) * (b - a + 1)) >> 1);

        for (i = a; i < b; i++){
            w = m / primes[i];
            lim = Lehmer(sqrt(w)), res -= Lehmer(w);

            if (i <= c){
                for (j = i; j < lim; j++){
                    res += j;
                    res -= Lehmer(w / primes[j]);
                }
            }
        }

        return res;
    }

    #define LEGENDRE -111
    #define LEHMER 666

    void random_exec(int lim, int flag){
        int i, x, y, z;
        int* val = new int[lim];

        mt19937 generator(time(0));
        tr1::uniform_int <int> random(1e9, 2e9);
        for (int i = 0; i < lim; i++) val[i] = random(generator);

        clock_t start = 0;
        for (i = 0; i < lim; i++){
            x = val[i];
            if (flag == LEGENDRE) y = Legendre(x);
```

```
            else if (flag == LEHMER) y = Lehmer(x);
        }

        printf("%0.3f s\n", (clock() - start) / (1.0 * CLOCKS_PER_SEC));
        delete[] val;
    }
}

int main(){
    pcf::init();
    pcf::random_exec(100, LEHMER);
    return 0;
}
```

Radix Sort (Unrolled with Bucket Arrays).c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>

#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int n;
unsigned int bucket[4][0x100], ar[MAX + 10], temp[MAX + 10];

void radix_sort(unsigned int* ar, int n){ /// hash = 639510
    int i;
    clr(bucket);
    for (i = 0; i < n; i++){
        bucket[0][ar[i] & 0xFF]++;
        bucket[1][(ar[i] >> 8) & 0xFF]++;
        bucket[2][(ar[i] >> 16) & 0xFF]++;
        bucket[3][(ar[i] >> 24) & 0xFF]++;
    }
    for (i = 1; i < 0x100; i++){
        bucket[0][i] += bucket[0][i - 1];
        bucket[1][i] += bucket[1][i - 1];
        bucket[2][i] += bucket[2][i - 1];
        bucket[3][i] += bucket[3][i - 1];
    }

    for (i = n - 1; i >= 0; i--) temp[--bucket[0][ar[i] & 0xFF]] = ar[i];
    for (i = n - 1; i >= 0; i--) ar[--bucket[1][(temp[i] >> 8) & 0xFF]] =
temp[i];
    for (i = n - 1; i >= 0; i--) temp[--bucket[2][(ar[i] >> 16) & 0xFF]] =
ar[i];
    for (i = n - 1; i >= 0; i--) ar[--bucket[3][(temp[i] >> 24) & 0xFF]] =
temp[i];
}

int main(){
```

```
   n = MAX;
   srand(time(0));
   int i, j, k, x, y;

   for (i = 0; i < n; i++) ar[i] = (rand() * rand());
   clock_t start = clock();
   radix_sort(ar, n);
   printf("%0.5f s\n", (clock() - start) / (1.0 * CLOCKS_PER_SEC));
   return 0;
}
```

Radix Sort (Unrolled).c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>

#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int n;
unsigned int bucket[0x100], ar[MAX + 10], temp[MAX + 10];

void radix_sort(unsigned int* ar, int n){
   int i;
   clr(bucket);
   for (i = 0; i < n; i++) bucket[ar[i] & 0xFF]++;
   for (i = 1; i < 0x100; i++) bucket[i] += bucket[i - 1];
   for (i = n - 1; i >= 0; i--) temp[--bucket[ar[i] & 0xFF]] = ar[i];
   clr(bucket);
   for (i = 0; i < n; i++) bucket[(temp[i] >> 8) & 0xFF]++;
   for (i = 1; i < 0x100; i++) bucket[i] += bucket[i - 1];
   for (i = n - 1; i >= 0; i--) ar[--bucket[(temp[i] >> 8) & 0xFF]] =
temp[i];
   clr(bucket);
   for (i = 0; i < n; i++) bucket[(ar[i] >> 16) & 0xFF]++;
   for (i = 1; i < 0x100; i++) bucket[i] += bucket[i - 1];
   for (i = n - 1; i >= 0; i--) temp[--bucket[(ar[i] >> 16) & 0xFF]] =
ar[i];
   clr(bucket);
   for (i = 0; i < n; i++) bucket[(temp[i] >> 24) & 0xFF]++;
   for (i = 1; i < 0x100; i++) bucket[i] += bucket[i - 1];
   for (i = n - 1; i >= 0; i--) ar[--bucket[(temp[i] >> 24) & 0xFF]] =
temp[i];
}

int main(){
   n = MAX;
   srand(time(0));
   int i, j, k, x, y;

   for (i = 0; i < n; i++) ar[i] = (rand() * rand());
```

```
   clock_t start = clock();
   radix_sort(ar, n);
   printf("%0.5f s\n", (clock() - start) / (1.0 * CLOCKS_PER_SEC));
   return 0;
}
```

Radix Sort uint_64 (Unrolled with Bucket Arrays).c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>

#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int n, bucket[8][0x100];
unsigned long long ar[MAX + 10], temp[MAX + 10];

void radix_sort(unsigned long long* ar, int n){
   int i;
   clr(bucket);
   for (i = 0; i < n; i++){
       bucket[0][ar[i] & 0xFF]++, bucket[1][(ar[i] >> 8) & 0xFF]++;
       bucket[2][(ar[i] >> 16) & 0xFF]++, bucket[3][(ar[i] >> 24) &
0xFF]++;
       bucket[4][(ar[i] >> 32) & 0xFF]++, bucket[5][(ar[i] >> 40) &
0xFF]++;
       bucket[6][(ar[i] >> 48) & 0xFF]++, bucket[7][(ar[i] >> 56) &
0xFF]++;
   }
   for (i = 1; i < 0x100; i++){
       bucket[0][i] += bucket[0][i - 1], bucket[1][i] += bucket[1][i -
1];
       bucket[2][i] += bucket[2][i - 1], bucket[3][i] += bucket[3][i -
1];
       bucket[4][i] += bucket[4][i - 1], bucket[5][i] += bucket[5][i -
1];
       bucket[6][i] += bucket[6][i - 1], bucket[7][i] += bucket[7][i -
1];
   }

   for (i = n - 1; i >= 0; i--) temp[--bucket[0][ar[i] & 0xFF]] = ar[i];
   for (i = n - 1; i >= 0; i--) ar[--bucket[1][(temp[i] >> 8) & 0xFF]] =
temp[i];
   for (i = n - 1; i >= 0; i--) temp[--bucket[2][(ar[i] >> 16) & 0xFF]] =
ar[i];
   for (i = n - 1; i >= 0; i--) ar[--bucket[3][(temp[i] >> 24) & 0xFF]] =
temp[i];
   for (i = n - 1; i >= 0; i--) temp[--bucket[4][(ar[i] >> 32) & 0xFF]] =
ar[i];
   for (i = n - 1; i >= 0; i--) ar[--bucket[5][(temp[i] >> 40) & 0xFF]] =
temp[i];
```

```
   for (i = n - 1; i >= 0; i--) temp[--bucket[6][(ar[i] >> 48) & 0xFF]] =
ar[i];
   for (i = n - 1; i >= 0; i--) ar[--bucket[7][(temp[i] >> 56) & 0xFF]] =
temp[i];
}

int main(){
   n = MAX;
   srand(time(0));
   int i, j, k, x, y;

   for (i = 0; i < n; i++){
       long long xx = (rand() * rand());
       long long xy = (rand() * rand());
       ar[i] = (xx * xy);
   }

   clock_t start = clock();
   radix_sort(ar, n);
   for (i = 1; i < n; i++){
       if (ar[i] < ar[i - 1]) puts("YO");
   }
   printf("%0.5f s\n", (clock() - start) / (1.0 * CLOCKS_PER_SEC));
   return 0;
}
```

Radix Sort.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>

#define CHUNK 8
#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int n;
unsigned int bucket[1 << CHUNK], ar[MAX + 10], temp[MAX + 10];

void radix_sort(unsigned int* ar, int n){
   int i, j, x;
   const int bitmask = (1 << CHUNK) - 1;

   for (i = 0; i < 32; i += CHUNK){
       clr(bucket);
       for (j = 0; j < n; j++) bucket[(ar[j] >> i) & bitmask]++;
       for (j = 1; j <= bitmask; j++){
           bucket[j] += bucket[j - 1];
       }

       for (j = n - 1; j >= 0; j--) temp[--bucket[(ar[j] >> i) &
bitmask]] = ar[j];
```

```
        for (j = 0; j < n; j++) ar[j] = temp[j];
    }
}

int main(){
    n = MAX;
    srand(time(0));
    int i, j, k, x, y;

    for (i = 0; i < n; i++) ar[i] = (rand() * rand());
    clock_t start = clock();
    radix_sort(ar, n);
    for (i = 0; (i + 1) < n; i++){
        if (ar[i] > ar[i + 1]) puts("YO");
    }
    printf("%0.5f s\n", (clock() - start) / (1.0 * CLOCKS_PER_SEC));
    return 0;
}
```

Random Prime Generator.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

bool isPrime(uint64_t p){
    for (uint64_t i = 2; (i * i) <= p; i++){
        if ((p % i) == 0) return false;
    }
    return true;
}

uint64_t RandomPrime(uint64_t a, uint64_t b){
    mt19937_64 generator(time(0));
    uniform_int_distribution<unsigned long long> random(a, b);

    uint64_t p = random(generator);
    while (!isPrime(p)) p++;
    return p;
}

int main(){
    uint64_t res = RandomPrime(1e9 + 9e8 , 2e9);
    dbg(res);
    return 0;
}
```

Range Factorization.c

```
#include <stdio.h>
```

```
#include <string.h>
#include <stdbool.h>

#define SQR 7071
#define MAX 50000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int L[MAX];
short P[MAX];

/*******************************************************
* P[0] = P[1] = 1, P[p] = 0 if p is a prime
* Otherwise, P[i] = the smallest prime factor if i
* L[0] = L[1] = 1, L[i] = the largest prime factor of i
*******************************************************/

void Generate(){
   int i, j, d, x;

   P[0] = P[1] = L[0] = L[1] = 1;
   for (i = 4; i < MAX; i++, i++) P[i] = 2;

   for (i = 3; i < SQR; i++, i++){
       if (!P[i]){
           d = i << 1;
           for (j = (i * i); j < MAX; j += d){
               if (!P[j]) P[j] = i;
           }
       }
   }

   for (i = 2; i < MAX; i++){
       if (!P[i]) L[i] = i;
       else{
           L[i] = P[i];
           x = L[i /P[i]];
           if (x > L[i]) L[i] = x;
       }
   }
}

int main(){
}
```

Rolling Hash.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
```

```
const unsigned long long base = 1968647011ULL;

int n, ar[MAX];
unsigned long long P[MAX];

void RollingHash(int len, bool gen){
   int i, j;
   unsigned long long h = 0, x;

   if (gen){
       P[0] = 1ULL;
       for (i = 1; i < MAX; i++) P[i] = (P[i - 1] * base);
   }

   for (i = 0; i < len; i++) h = (h * base) + ar[i];
   for (i = 0; (i + len) <= n; i++){
       /* h contains required hash value now */

       x = (h - (P[len - 1] * ar[i]));
       h = (x * base) + ar[i + len];
   }
   return 0;
}

int main(){
}
```

Rope.cpp

```
--------------------------------------------------
#include <ext/rope>
#include <bits/stdtr1c++.h>

#define MAX 50010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;
using namespace __gnu_cxx;

rope <char> R[MAX];
int d = 0, ye = 0, vnow = 0;
char str[105], out[10000010];

int main(){
   int n, i, j, k, v, p, c, x, flag;

   while (scanf("%d", &n) != EOF){
       d = 0, vnow = 0;
       while (n--){
           scanf("%d", &flag);

           if (flag == 1){
```

```
                scanf("%d %s", &p, str);
                p -= d, vnow++;
                R[vnow] = R[vnow - 1];
                R[vnow].insert(p, str); /// Insert string str after
position p
            }

            if (flag == 2){
                scanf("%d %d", &p, &c);
                p -= d, c -= d, vnow++;
                R[vnow] = R[vnow - 1];
                R[vnow].erase(p - 1, c); /// Remove c characters starting
at position p
            }

            if (flag == 3){
                scanf("%d %d %d", &v, &p, &c); /// Print c characters
starting at position p in version v

                v -= d, p -= d, c -= d;
                rope <char> sub = R[v].substr(p - 1, c); /// Get the
substring of c characters from position p in version v
                for (auto it: sub){
                    out[ye++] = it;
                    if (it == 'c') d++;
                }
                out[ye++] = 10;
            }
        }
    }

    fwrite(out, 1, ye, stdout);
    return 0;
}
```

SCC.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 200010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define neg(x) ((x) <= n ? (x + n) : (x - n))

using namespace std;

int n;
bool visited[MAX], val[MAX];
int l, cmp, ar[MAX], num[MAX];
vector <int> adj[MAX], rev[MAX];

void topsort(int i){
    visited[i] = true;
```

```
    int j, x, len = adj[i].size();

    for (j = 0; j < len; j++){
        x = adj[i][j];
        if (!visited[x]) topsort(x);
    }
    ar[l++] = i;
}

void dfs(int i){
    num[i] = cmp;
    visited[i] = true;
    int j, x, len = rev[i].size();

    for (j = 0; j < len; j++){
        x = rev[i][j];
        if (!visited[x]) dfs(x);
    }
}

void SCC(){
    int i, j, x;
    l = 0, cmp = 0;

    clr(visited);
    for (i = 0; i < n; i++){
        if (!visited[i]) topsort(i);
    }

    clr(visited);
    for (i = l - 1; i >= 0; i--){
        x = ar[i];
        if (!visited[x]){
            cmp++;
            dfs(x);
        }
    }
}

int main(){
    int T = 0, t, q, a, b;

    scanf("%d", &t);
    while (t--){
        clr(adj), clr(rev);
        scanf("%d %d", &n, &q);

        while (q--){
            scanf("%d %d", &a, &b);
            adj[a].push_back(b);
            rev[b].push_back(a);
        }

        SCC();
```

```
    }
    return 0;
}
```

Segment Line Point.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

struct Point{
    long long x, y;

    Point(){
    }

    Point(long long xi, long long yi){
        x = xi, y = yi;
    }
};

struct Segment{
    struct Point P1, P2;

    Segment(){
    }

    Segment(struct Point P1i, struct Point P2i){
        P1 = P1i, P2 = P2i;
    }
};

/// Returns 0 if ABC is collinear, positive if ABC is a left turn,
negative if ABC is a right turn
long long ccw(struct Point A, struct Point B, struct Point C){
    return ((B.x - A.x) * (C.y - A.y)) - ((C.x - A.x) * (B.y - A.y));
}

/// Returns the shortest distance from Segment S to Point P
double dis(struct Segment S, struct Point P){
    double p, xx, yy;
    long long x = P.x, y = P.y, x1 = S.P1.x, y1 = S.P1.y, x2 = S.P2.x, y2
= S.P2.y;
    long long a = x - x1, b = y - y1, c = x2 - x1, d = y2 - y1, dot = (a *
c) + (b * d), len = (c * c) + (d * d);

    if ((dot < 0) || (x1 == x2 && y1 == y2)) xx = x1, yy = y1;
    else if (dot > len) xx = x2, yy = y2;
```

```
   else p = (1.0 * dot) / len, xx = x1 + (p * c), yy = y1 + (p * d);
   xx = -xx + x, yy = -yy + y;
   return sqrt((xx * xx) + (yy * yy));
}

/// Returns true if Point P lies on the Segment (both end-points
inclusive)
bool PointOnSeg(struct Segment S, struct Point P){
   long long x = P.x, y = P.y, x1 = S.P1.x, y1 = S.P1.y, x2 = S.P2.x, y2
= S.P2.y;
   long long a = x - x1, b = y - y1, c = x2 - x1, d = y2 - y1, dot = (a *
c) + (b * d), len = (c * c) + (d * d);

   if (x1 == x2 && y1 == y2) return (x1 == x && y1 == y);
   if (dot < 0 || dot > len) return false;
   return ((((x1 * len) + (dot * c)) == (x * len)) && (((y1 * len) + (dot
* d)) == (y * len)));
}

int main(){
   return 0;
}
```

Segment Tree (Lazy Propagation).c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 50010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int n, q;
int tree[MAX << 2], lazy[MAX << 2];

void propagate(int idx, int a, int b){
   if (lazy[idx]){
       int p = (idx << 1), q = p + 1;
       int c = (a + b) >> 1, d = c + 1;

       tree[idx] += (lazy[idx] * (b - a + 1));
       if (a != b){
           lazy[p] += lazy[idx];
           lazy[q] += lazy[idx];
       }
       lazy[idx] = 0;
   }
}

void update(int* tree, int* lazy, int idx, int a, int b, int l, int r,
int x){
   int p = (idx << 1), q = p + 1;
   int c = (a + b) >> 1, d = c + 1;
```

```
    if (a == l && b == r) lazy[idx] += x;
    propagate(idx, a, b);
    if (a == l && b == r) return;

    if (r <= c){
        propagate(q, d, b);
        update(tree, lazy, p, a, c, l, r, x);
    }
    else if (l >= d){
        propagate(p, a, c);
        update(tree, lazy, q, d, b, l, r, x);
    }
    else{
        update(tree, lazy, p, a, c, l, c, x);
        update(tree, lazy, q, d, b, d, r, x);
    }

    tree[idx] = tree[p] + tree[q];
}

int query(int* tree, int* lazy, int idx, int a, int b, int l, int r){
    int p = (idx << 1), q = p + 1;
    int c = (a + b) >> 1, d = c + 1;

    propagate(idx, a, b);
    if (a == l && b == r) return tree[idx];
    if (r <= c) return query(tree, lazy, p, a, c, l, r);
    else if (l >= d) return query(tree, lazy, q, d, b, l, r);
    else{
        int x = query(tree, lazy, p, a, c, l, c);
        int y = query(tree, lazy, q, d, b, d, r);
        return (x + y);
    }
}

void update_range(int l, int r, int v){
    update(tree, lazy, 1, 1, n, l, r, v);
}

int query_range(int l, int r){
    return query(tree, lazy, 1, 1, n, l, r);
}

int main(){
}

Segment Tree Generic.cpp
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 200010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
```

```
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

int n, q;
long long ar[MAX];

/// Look thoroughly for a change
struct SegTree{
   int n;

   struct Node{
       long long a, d, inc, val;

       Node(){
       }

       Node(vector <int> ar){
           a = ar[0], d = ar[1], inc = ar[2], val = ar[3];
       }
   } tree[MAX << 2], nil = Node({0, 0, 0, 0}); /// If runtime error,
replace array with vector
   /// nil should be chosen properly


   SegTree(){
   }

   SegTree(int m){
       n = m;
       for (int i = 0; i <= (n << 2); i++) tree[i] = nil; /// Tree
initialization
   }

   long long F(long long a, long long d, long long inc, long long n){
       long long res = 0;
       while (n--){
           res += a;
           a += d;
           d += inc;
       }
       return res;
   }

   /// Add to information to left(ld) and right(rd) child ld and rd using
information from data
   /// l = number of elements to be updated in ld, r = number of elements
to be updated in rd
   void split(int l, int r, struct Node& data, struct Node& ld, struct
Node& rd){
       ld.a += data.a, ld.d += data.d, ld.inc += data.inc;
       long long m = l + 1;
       rd.a += ( data.a + ((m - 1) * data.d) + ((((m - 1) * (m - 2)) >>
1) * data.inc) );
```

```
        rd.d += (data.d + (data.inc * (m - 1))), rd.inc += data.inc;
    }

    /// Update tree node during segment tree update
    void update_node(int idx, int a, int b, int l, int r, struct Node&
data){
        tree[idx].a += data.a, tree[idx].d += data.d, tree[idx].inc +=
data.inc;
    }

    /// Update parent from left and right child
    void update_parent(int idx, int p, int q){
        tree[idx].val = tree[p].val + tree[q].val;
    }

    void propagate(int idx, int a, int b, int counter){
        if (tree[idx].a){
            int p = (idx << 1), q = p | 1;
            int c = (a + b) >> 1, d = c + 1;
            tree[idx].val = tree[idx].val + F(tree[idx].a, tree[idx].d,
tree[idx].inc, b - a + 1); /// Lazy update node

            if (a != b){
                split((c - a + 1), (d - b + 1), tree[idx], tree[p],
tree[q]); /// Propagate lazy values to left and right child

                if (counter--){
                    propagate(p, a, c, counter);
                    propagate(q, d, b, counter);
                }
            }
            tree[idx].a = tree[idx].d = tree[idx].inc = 0; /// Reset lazy
values
        }
    }

    void update(int idx, int a, int b, int l, int r, struct Node& data){
        if (a == l && b == r){
            update_node(idx, a, b, l, r, data); /// Update whole segment
            propagate(idx, a, b, 1);
            return;
        }
        propagate(idx, a, b, 1);


        int p = (idx << 1), q = p | 1;
        int c = (a + b) >> 1, d = c + 1;

        if (r <= c){
            propagate(q, d, b, 1);
            update(p, a, c, l, r, data);
        }
        else if (l >= d){
            propagate(p, a, c, 1);
```

```
            update(q, d, b, l, r, data);
        }
        else{
            struct Node ld = nil, rd = nil;
            split((c - l + 1), (r - d + 1), data, ld, rd);
            update(p, a, c, l, c, ld);
            update(q, d, b, d, r, rd);
        }

        update_parent(idx, p, q); /// Update current node using
information from left and right child
    }

    void update(int l, int r){
        struct Node data = Node({2, 4, 2, 0}); /// Change accordingly
        update(1, 1, n, l, r, data);
    }

    long long query(int idx, int a, int b, int l, int r){
        int p = (idx << 1), q = p + 1;
        int c = (a + b) >> 1, d = c + 1;

        propagate(idx, a, b, 1);
        if (a == l && b == r) return tree[idx].val;
        if (r <= c) return query(p, a, c, l, r);
        else if (l >= d) return query(q, d, b, l, r);
        else{
            long long x = query(p, a, c, l, c);
            long long y = query(q, d, b, d, r);
            return (x + y); /// Change accordingly
        }
    }

    long long query(int l, int r){
        return query(1, 1, n, l, r);
    }
};

void run(){
    int flag, i, l, r;
    struct SegTree S = SegTree(n);

    while (q--){
        scanf("%d %d %d", &flag, &l, &r);
        if (flag == 1){
            S.update(l, r);
        }
        else printf("%lld\n", S.query(l, r));
    }
}

int main(){
    while (scanf("%d %d", &n, &q) != EOF){
        run();
```

```
    }
    return 0;
}
```

Segmented Sieve.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 1000010

#define BASE_SQR 216
#define BASE_LEN 10010
#define BASE_MAX 46656

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define chkbit(ar, i) (((ar[(i) >> 6]) & (1 << (((i) >> 1) & 31))))
#define setbit(ar, i) (((ar[(i) >> 6]) |= (1 << (((i) >> 1) & 31))))

int p, primes[BASE_LEN];
unsigned int base[(BASE_MAX >> 6) + 5], isprime[(MAX >> 6) + 5];

void Sieve(){
   clr(base);
   int i, j, k;

   for (i = 3; i < BASE_SQR; i++, i++){
       if (!chkbit(base, i)){
           k = i << 1;
           for (j = (i * i); j < BASE_MAX; j += k){
               setbit(base, j);
           }
       }
   }

   p = 0;
   for (i = 3; i < BASE_MAX; i++, i++){
       if (!chkbit(base, i)){
           primes[p++] = i;
       }
   }
}

int SegmentedSieve(long long a, long long b){
   long long j, k, x;
   int i, d, counter = 0;

   if (a <= 2 && 2 <= b) counter = 1; /// 2 is counted separately if in
range
   if (!(a & 1)) a++;
   if (!(b & 1)) b--;
   if (a > b) return counter;
```

```
    clr(isprime);
    for (i = 0; i < p; i++){
        x = primes[i];
        if ((x * x) > b) break;

        k = x << 1;
        j = x * ((a + x - 1) / x);
        if (!(j & 1)) j += x;
        else if (j == x) j += k;

        while (j <= b){
            setbit(isprime, j - a);
            j += k;
        }
    }

    /// Other primes in the range except 2 are added here
    d = (b - a + 1);
    for (i = 0; i < d; i++, i++){
        if (!chkbit(isprime, i) && (a + i) != 1) counter++;
    }

    return counter;
}

int main(){
    Sieve();
    int T = 0, t, i, j, a, b;

    scanf("%d", &t);
    while (t--){
        scanf("%d %d", &a, &b);
        printf("Case %d: %d\n", ++T, SegmentedSieve(a, b));
    }
    return 0;
}
```

Shank_s Factorization Algorithm OP 2.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 1000010
#define isprime(x) prm::miller_rabin(x)
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

namespace prm{
    bitset <MAX> flag;
    long double op = 0.0;
```

```
int p = 0, prime[78777];
const int base[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};

void sieve(){
    int i, j, x;
    for (i = 3; i < MAX; i += 2) flag[i] = true;
    for (i = 3, flag[2] = true; (i * i) < MAX; i += 2){
        if (flag[i]){
            for (j = (i * i), x = i << 1; j < MAX; j += x){
                flag[j] = false;
            }
        }
    }

    for (i = 2; i < MAX; i++){
        if (flag[i]) prime[p++] = i;
    }
}

void init(){
    if (!flag[2]) sieve();
}

inline long long mul(long long a, long long b, long long MOD){
   if ((MOD < 3037000500LL)) return ((a * b) % MOD);
   long double res = a;
   res *= b;
   long long c = (long long)(res * op);
   a *= b;
   a -= c * MOD;
   if (a >= MOD) a -= MOD;
   if (a < 0) a += MOD;
   return a;
}

inline long long expo(long long x, long long n, long long m){
    long long res = 1;

    while (n){
        if (n & 1) res = mul(res, x, m);
        x = mul(x, x, m);
        n >>= 1;
    }

    return (res % m);
}

inline bool miller_rabin(long long p){
    if (p < MAX) return flag[p];
    if ((p + 1) & 1) return false;
    for (int i = 1; i < 10; i++){
        if (!(p % prime[i])) return false;
    }
```

```
        long long a, m, x, s = p - 1, y = p - 1;
        op = (long double)1 / p, s = s >> __builtin_ctzll(s);

        for (int i = 0; i < 7; i++) {
            x = s, a = (base[i] % y) + 1;
            m = expo(a, x, p);
            while ((x != y) && (m != 1) && (m != y)) m = mul(m, m, p), x
<<= 1;
            if ((m != y) && !(x & 1)) return false;
        }
        return true;
    }

    inline long long countdiv(long long n){
        int i, j, c;
        long long x, res = 1;
        for (i = 0; i < p; i++){
            x = prime[i];
            if ((x * x * x) > n) break;

            c = 1;
            while (!(n % x)) c++, n /= x;
            if (c > 1){
                res *= c;
                if (miller_rabin(n)){
                    res <<= 1, n = 1;
                    break;
                }
            }
        }

        if (n > 1 && miller_rabin(n)) res <<= 1;
        else if (n > 1) {
            x = sqrt((long double)0.975 + n);
            if ((x * x) == n && miller_rabin(x)) res *= 3;
            else res <<= 2;
        }
        return res;
    }

    inline long long isqrt(long long x){
        return (long long)sqrtl((long double)0.975 + x);
    }

    inline bool is_square(long long x){
        long long y = isqrt(x);
        return (y * y == x);
    }

    /// Shank's Factorization Algorithm (SQUFOF)
    /// Complexity: O(N^(1/4)), (N must be greater than 1 and not a prime)
    /// Algorithms terminates successfully if all required multiplications
do not overflow, otherwise throws std::overflow_error
```

```
    /// All numbers less than 2^58 are guaranteed to find a factor
successfully

    inline long long shanks(long long n){
        auto find_factor = [n](const long long k)->long long{
            long long p, q, b, x, old_p, old_q, sqrt_q, sqrt_kn;

            p = sqrt_kn = isqrt(k * n);
            q = k * n - p * p, old_p = old_q = 1;
            if (q == 0) return (k == 1) ? p : 1;

            auto update = [&]{
                old_p = p;
                b = (sqrt_kn + old_p) / q, p = b * q - old_p;
                x = q, q = old_q + b * (old_p - p), old_q = x;
            };
            for (int i = 1; (i & 1) || !is_square(q); i++) update();

            sqrt_q = isqrt(q), b = (sqrt_kn - p) / sqrt_q;
            p = b * sqrt_q + p, old_q = sqrt_q, q = (k * n - p * p) /
old_q;

            do{
                update();
            } while (p != old_p);
            return __gcd(n, p);
        };

        const long long lim = std::numeric_limits <long long>::max() / n;
        for (long long k = 1; k <= lim; k++){
            long long f = find_factor(k);
            if (f != 1 && f != n) return f;
        }

        throw std::overflow_error("Shanks overflow error!\n");
        return -1;
    }

    inline void factorize(long long n, vector <long long>& res){
        if (n == 1) return;
        if (isprime(n)){
            res.push_back(n);
            return;
        }

        long long f = shanks(n);
        factorize(f, res);
        factorize(n / f, res);
    }

    inline vector <long long> factorize(long long n){
        vector <long long> res;
        for (int i = 0; i < p; i++){
```

```
            if ((long long)prime[i] * prime[i] * prime[i] * prime[i] > n)
break;
            while (n % prime[i] == 0){
                n /= prime[i];
                res.push_back(prime[i]);
            }
        }
        factorize(n, res);

        sort(res.begin(), res.end());
        return res;
    }
}

int main(){
    prm::init();
    vector <long long> v = prm::factorize(12 * 31 * 7 * 997 * 17 *
2147483647LL);
    for (int i = 0; i < v.size(); i++) dbg(v[i]);
}
```

Shank_s Factorization Algorithm OP.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 1000010
#define isprime(x) prm::miller_rabin(x)
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

namespace prm{
    bitset <MAX> flag;
    long double op = 0.0;
    int p = 0, prime[78777];
    const int base[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};

    void Sieve(){
        int i, j, x;
        for (i = 3; i < MAX; i += 2) flag[i] = true;
        for (i = 3, flag[2] = true; (i * i) < MAX; i += 2){
            if (flag[i]){
                for (j = (i * i), x = i << 1; j < MAX; j += x){
                    flag[j] = false;
                }
            }
        }

        for (i = 2; i < MAX; i++){
            if (flag[i]) prime[p++] = i;
        }
```

```
    }

    void init(){
        if (!flag[2]) Sieve();
    }

    inline long long mul(long long a, long long b, long long MOD){
       if ((MOD < 3037000500LL)) return ((a * b) % MOD);
       long double res = a;
       res *= b;
       long long c = (long long)(res * op);
       a *= b;
       a -= c * MOD;
       if (a >= MOD) a -= MOD;
       if (a < 0) a += MOD;
       return a;
    }

    inline long long expo(long long x, long long n, long long m){
        long long res = 1;

        while (n){
            if (n & 1) res = mul(res, x, m);
            x = mul(x, x, m);
            n >>= 1;
        }

        return (res % m);
    }

    inline bool miller_rabin(long long p){
        if (p < MAX) return flag[p];
        if ((p + 1) & 1) return false;
        for (int i = 1; i < 10; i++){ /// basic iterations
            if (!(p % prime[i])) return false;
        }

        long long a, m, x, s = p - 1, y = p - 1;
        op = (long double)1 / p, s = s >> __builtin_ctzll(s);

        for (int i = 0; i < 7; i++) {
            x = s, a = (base[i] % y) + 1;
            m = expo(a, x, p);
            while ((x != y) && (m != 1) && (m != y)) m = mul(m, m, p), x
<<= 1;
            if ((m != y) && !(x & 1)) return false;
        }
        return true;
    }

    /// n cannot be a prime or a square number
    /// Returns one factor of n, k = base multiplier for convergence
    /// Returns 1 on failure, try with next value of k. Returns 0 if not
possible to factor due to overflow
```

```
inline unsigned long long shanks(unsigned long long n, int k){
    int i, j, u, v;
    unsigned long long m, b, x, g, s, P[2], Q[2];

    m = n * k;
    if ((m / n) != k) return 0;
    s = sqrt((long double)m + 0.975);
    Q[0] = 1, P[0] = s, Q[1] = m - (P[0] * P[0]);

    for (i = 1; ;i++){
        u = i & 1, v = (i + 1) & 1;
        if (v){
            x = sqrt((long double)Q[u] + 0.95);
            if ((x * x) == Q[u]){
                b = (s - P[v]) / x;
                P[0] = (b * x) + P[v];
                Q[0] = x, Q[1] = (m - P[0] * P[0]) / Q[0];
                break;
            }
        }

        b = (s + P[v]) / Q[u];
        P[u] = (b * Q[u]) - P[v];
        Q[v] = Q[v] + (b * (P[v] - P[u]));
    }


    for (i = 1; ;i++){
        u = i & 1, v = (i + 1) & 1;
        b = (s + P[v]) / Q[u];
        P[u] = (b * Q[u]) - P[v];
        Q[v] = Q[v] + (b * (P[v] - P[u]));
        if (P[u] == P[v]) break;
    }

    g = __gcd(n, P[u]);
    return ((g == 1 || g == n) ? 1 : g);
}

inline long long countdiv(long long n){
    int i, j, c;
    long long x, res = 1;
    for (i = 0; i < p; i++){
        x = prime[i];
        if ((x * x * x) > n) break;

        c = 1;
        while (!(n % x)) c++, n /= x;
        if (c > 1){
            res *= c;
            if (miller_rabin(n)){
                res <<= 1, n = 1;
                break;
            }
```

```
        }
    }

    if (n > 1 && miller_rabin(n)) res <<= 1;
    else if (n > 1) {
        x = sqrt((long double)0.975 + n);
        if ((x * x) == n && miller_rabin(x)) res *= 3;
        else res <<= 2;
    }

    return res;
}

inline vector <long long> factorize(long long n){ /// Works well for n
<= 10^17
    long long x;
    vector <long long> factors;

    for (int i = 0, c = 0; i < p; i++){
        c = 0, x = prime[i];
        if ((x * x * x) > n) break;

        while (!(n % x)){
            n /= x, c++;
            factors.push_back(x);
        }
        if (c && miller_rabin(n)){
            factors.push_back(n);
            n = 1;
            break;
        }
    }

    if (n > 1 && miller_rabin(n)) factors.push_back(n);
    else if (n > 1) {
        x = sqrt((long double)0.975 + n);
        if ((x * x) == n && miller_rabin(x)){
            factors.push_back(x);
            factors.push_back(x);
        }
        else{
            for (int i = 1; ;i++){
                x = shanks(n, i); /// Overflow problem for some
numbers > 10^17, since maximum i required can be around 32
                if (x != 1 && x != n){
                    factors.push_back(x);
                    factors.push_back(n / x);
                    break;
                }
            }
        }
    }

    sort(factors.begin(), factors.end());
```

```
        return factors;
    }
}

int main(){
    prm::init();
}
```

Shank_s Factorization Algorithm.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 1000010
#define isprime(x) prm::miller_rabin(x)
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

namespace prm{
    bitset <MAX> flag;
    long double op = 0.0;
    int p = 0, prime[78777];
    const int base[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};

    void Sieve(){
        int i, j, x;
        for (i = 3; i < MAX; i += 2) flag[i] = true;
        for (i = 3, flag[2] = true; (i * i) < MAX; i += 2){
            if (flag[i]){
                for (j = (i * i), x = i << 1; j < MAX; j += x){
                    flag[j] = false;
                }
            }
        }

        for (i = 2; i < MAX; i++){
            if (flag[i]) prime[p++] = i;
        }
    }

    void init(){
        if (!flag[2]) Sieve();
    }

    inline long long mul(long long a, long long b, long long MOD){
       if ((MOD < 3037000500LL)) return ((a * b) % MOD);
       long double res = a;
       res *= b;
       long long c = (long long)(res * op);
       a *= b;
       a -= c * MOD;
```

```
        if (a >= MOD) a -= MOD;
        if (a < 0) a += MOD;
        return a;
    }

    inline long long expo(long long x, long long n, long long m){
        long long res = 1;

        while (n){
            if (n & 1) res = mul(res, x, m);
            x = mul(x, x, m);
            n >>= 1;
        }

        return (res % m);
    }

    inline bool miller_rabin(long long p){
        if (p < MAX) return flag[p];
        if ((p + 1) & 1) return false;
        for (int i = 1; i < 10; i++){ /// basic iterations
            if (!(p % prime[i])) return false;
        }

        long long a, m, x, s = p - 1, y = p - 1;
        op = (long double)1 / p, s = s >> __builtin_ctzll(s);

        for (int i = 0; i < 7; i++) {
            x = s, a = (base[i] % y) + 1;
            m = expo(a, x, p);
            while ((x != y) && (m != 1) && (m != y)) m = mul(m, m, p), x
<<= 1;
            if ((m != y) && !(x & 1)) return false;
        }
        return true;
    }

    /// n cannot be a prime or a square number
    /// Returns one factor of n, k = base multiplier for convergence
    /// Returns 1 on failure, try with next value of k. Returns 0 if not
possible to factor due to overflow
    inline unsigned long long shanks(unsigned long long n, int k){
        int i, j, u, v;
        unsigned long long m, b, x, g, s, P[2], Q[2];

        m = n * k;
        if ((m / n) != k) return 0;
        s = sqrt((long double)m + 0.975);
        Q[0] = 1, P[0] = s, Q[1] = m - (P[0] * P[0]);

        for (i = 1; ;i++){
            u = i & 1, v = (i + 1) & 1;
            if (v){
                x = sqrt((long double)Q[u] + 0.95);
```

```
            if ((x * x) == Q[u]){
                b = (s - P[v]) / x;
                P[0] = (b * x) + P[v];
                Q[0] = x, Q[1] = (m - P[0] * P[0]) / Q[0];
                break;
            }
        }

        b = (s + P[v]) / Q[u];
        P[u] = (b * Q[u]) - P[v];
        Q[v] = Q[v] + (b * (P[v] - P[u]));
    }


    for (i = 1; ;i++){
        u = i & 1, v = (i + 1) & 1;
        b = (s + P[v]) / Q[u];
        P[u] = (b * Q[u]) - P[v];
        Q[v] = Q[v] + (b * (P[v] - P[u]));
        if (P[u] == P[v]) break;
    }

    g = __gcd(n, P[u]);
    return ((g == 1 || g == n) ? 1 : g);
}

inline long long countdiv(long long n){
    int i, j, c;
    long long x, res = 1;
    for (i = 0; i < p; i++){
        x = prime[i];
        if ((x * x * x) > n) break;

        c = 1;
        while (!(n % x)) c++, n /= x;
        res *= c;
    }

    if (miller_rabin(n)) res <<= 1;
    else if (n > 1) {
        x = sqrt((long double)0.975 + n);
        if ((x * x) == n && miller_rabin(x)) res *= 3;
        else res <<= 2;
    }

    return res;
}

inline vector <long long> factorize(long long n){ /// Works well for n
<= 10^17
    long long x;
    vector <long long> factors;

    for (int i = 0; i < p; i++){
```

```
            x = prime[i];
            if ((x * x * x) > n) break;
            if (!(i & 1023) && miller_rabin(n)) break;

            while (!(n % x)){
                n /= x;
                factors.push_back(x);
            }
        }

        if (miller_rabin(n)) factors.push_back(n);
        else if (n > 1) {
            x = sqrt((long double)0.975 + n);
            if ((x * x) == n && miller_rabin(x)){
                factors.push_back(x);
                factors.push_back(x);
            }
            else{
                for (int i = 1; ;i++){
                    x = shanks(n, i); /// Overflow problem for some
numbers > 10^17, since maximum i required can be around 32
                    if (x != 1 && x != n){
                        factors.push_back(x);
                        factors.push_back(n / x);
                        break;
                    }
                }
            }
        }

        sort(factors.begin(), factors.end());
        return factors;
    }
}

int main(){
   prm::init();
}
```

Sieve(Bitmask).c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define LEN 78777
#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define chkbit(ar, i) (((ar[(i) >> 6]) & (1 << (((i) >> 1) & 31))))
#define setbit(ar, i) (((ar[(i) >> 6]) |= (1 << (((i) >> 1) & 31))))
#define isprime(x) (( (x) && ((x)&1) && (!chkbit(ar, (x)))) || ((x) ==
2))
```

```
int p, prime[LEN];
unsigned int ar[(MAX >> 6) + 5] = {0};

void Sieve(){
   int i, j, k;
   setbit(ar, 0), setbit(ar, 1);

   for (i = 3; (i * i) < MAX; i++, i++){
       if (!chkbit(ar, i)){
           k = i << 1;
           for (j = (i * i); j < MAX; j += k) setbit(ar, j);
       }
   }

   p = 0;
   prime[p++] = 2;
   for (i = 3; i < MAX; i++, i++){
       if (isprime(i)) prime[p++] = i;
   }
}

int main(){
   Sieve();
   printf("%d\n", p);
   int i;
   for (i = 0; i < 60; i++){
       if (isprime(i)) printf("%d\n", i);
   }
}
```

Sieve.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int p, P[MAX];
bool prime[MAX];

void Sieve(){
   clr(prime);
   int i, j, d;
   const int sqr = ;

   prime[2] = true;
   for (i = 3; i < MAX; i++, i++) prime[i] = true;

   for (i = 3; i < sqr;){
       d = i << 1;
       for (j = (i * i); j < MAX; j += d) prime[j] = false;
```

```
        i++, i++;
        while (!prime[i]) i++, i++;
    }

    p = 0;
    for (i = 0; i < MAX; i++){
        if (prime[i]) P[p++] = i;
    }
}

int main(){
    Sieve();
}
```

Simple Polygon From Points.c

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

struct Point{
    int idx, x, y, d;
    double theta;
};

int n;
struct Point ar[2010];
double centre_x, centre_y;

int compare(const void* a, const void* b){
    struct Point P1 = *(struct Point*)a;
    struct Point P2 = *(struct Point*)b;

    if (fabs(P1.theta - P2.theta) <= 1e-9){
        double d1 = ((centre_x - P1.x) * (centre_x - P1.x)) + ((centre_y -
P1.y) + (centre_y - P1.y));
        double d2 = ((centre_x - P2.x) * (centre_x - P2.x)) + ((centre_y -
P2.y) + (centre_y - P2.y));
        if (fabs(d1 - d2) <= 1e-9) return 0;
        else if (d1 < d2) return -1;
        else return +1;
    }
    else if (P1.theta < P2.theta) return +1;
    else return -1;
}

int main(){
    int t, line, i, j;
```

```
    scanf("%d", &t);
    for (line = 1; line <= t; line++){
        scanf("%d", &n);
        centre_x = 0.0, centre_y = 0.0;

        for (i = 0; i < n; i++){
            scanf("%d %d", &ar[i].x, &ar[i].y);
            ar[i].idx = i;
            centre_x += ar[i].x;
            centre_y += ar[i].y;
        }

        centre_x /= (1.0 * n), centre_y /= (1.0 * n);
        for (i = 0; i < n; i++) ar[i].theta = atan2((double)ar[i].y -
centre_y, (double)ar[i].x - centre_x);
        qsort(ar, n, sizeof(struct Point), compare);

        for (i = 0; i < n; i++){
            if (i != 0) putchar(32);
            printf("%d", ar[i].idx);
        }
        puts("");
    }
    return 0;
}
```

Simplex (Namespace + Long Double).cpp

```
#include <bits/stdtr1c++.h>

#define MAXC 1010
#define MAXV 1010
#define EPS 1e-13

#define MINIMIZE -1
#define MAXIMIZE +1
#define LESSEQ -1
#define EQUAL 0
#define GREATEQ 1
#define INFEASIBLE -1
#define UNBOUNDED 666

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/***
1.  Simplex Algorithm for Linear Programming
2.  Maximize or minimize f0*x0 + f1*x1 + f2*x2 + ... + fn-1*xn-1 subject
to some constraints
```

```
3.  Constraints are of the form, c0x0 + c1x1 + c2x2 + ... + cn-1xn-1 (<=
or >= or =) lim
4.  m is the number of constraints indexed from 1 to m, and n is the
number of variables indexed from 0 to n-1
5.  ar[0] contains the objective function f, and ar[1] to ar[m] contains
the constraints, ar[i][n] = lim_i
6.  It is assumed that all variables satisfies non-negativity constraint,
i.e, xi >= 0
7.  If non-negativity constraint is not desired for a variable x, replace
each occurrence
   by difference of two new variables r1 and r2 (where r1 >= 0 and r2 >=
0, handled automatically by simplex).
   That is, replace every x by r1 - r2 (Number of variables increases by
one, -x, +r1, +r2)
8.  solution_flag = INFEASIBLE if no solution is possible and UNBOUNDED
if no finite solution is possible
9.  Returns the maximum/minimum value of the linear equation satisfying
all constraints otherwise
10. After successful completion, val[] contains the values of x0, x1 ....
xn for the optimal value returned

*** If ABS(X) <= M in constraints, Replace with X <= M and -X <= M

*** Fractional LP:

   max/min
       3x1 + 2x2 + 4x3 + 6
       -------------------
       3x1 + 3x2 + 2x3 + 5

       s.t. 2x1 + 3x2 + 5x3  0

   Replace with:

   max/min
       3y1 + 2y2 + 4y3 + 6t

       s.t. 3y1 + 3y2 + 2y3 + 5t = 1
       2y1 + 3y2 + 53 - 23t  0

***/

namespace lp{
   long double val[MAXV], ar[MAXC][MAXV], mat[MAXC][MAXV];
   int m, n, solution_flag, minmax_flag, basis[MAXC], index[MAXV];

   /// nvars = number of variables, f = objective function, flag =
MINIMIZE or MAXIMIZE
   inline void init(int nvars, long double f[], int flag){
       solution_flag = 0;
       ar[0][nvars] = 0.0;
       m = 0, n = nvars, minmax_flag = flag;
       for (int i = 0; i < n; i++){
```

```
            ar[0][i] = f[i] * minmax_flag; /// Negating sign of objective
function when minimizing
        }
    }

    /// C[] = co-efficients of the constraints (LHS), lim = limit in RHS
    /// cmp = EQUAL for C[] = lim, LESSEQ for C[] <= lim, GREATEQ for C[]
>= lim
    inline void add_constraint(long double C[], long double lim, int cmp){
        m++, cmp *= -1;
        if (cmp == 0){
            for (int i = 0; i < n; i++) ar[m][i] = C[i];
            ar[m++][n] = lim;
            for (int i = 0; i < n; i++) ar[m][i] = -C[i];
            ar[m][n] = -lim;
        }
        else{
            for (int i = 0; i < n; i++) ar[m][i] = C[i] * cmp;
            ar[m][n] = lim * cmp;
        }
    }

    inline void init(){ /// Initialization
        for (int i = 0; i <= m; i++) basis[i] = -i;
        for (int j = 0; j <= n; j++){
            ar[0][j] = -ar[0][j], index[j] = j, val[j] = 0;
        }
    }

    inline void pivot(int m, int n, int a, int b){ /// Pivoting and
exchanging a non-basic variable with a basic variable
        for (int i = 0; i <= m; i++){
            if (i != a){
                for (int j = 0; j <= n; j++){
                    if (j != b){
                        ar[i][j] -= (ar[i][b] * ar[a][j]) / ar[a][b];
                    }
                }
            }
        }

        for (int j = 0; j <= n; j++){
            if (j != b) ar[a][j] /= ar[a][b];
        }
        for (int i = 0; i <= m; i++){
            if (i != a) ar[i][b] = -ar[i][b] / ar[a][b];
        }

        ar[a][b] = 1.0 / ar[a][b];
        swap(basis[a], index[b]);
    }

    inline long double solve(){ /// simplex core
        init();
```

```
        int i, j, k, l;
        for (; ;){
            for (i = 1, k = 1; i <= m; i++){
                if ((ar[i][n] < ar[k][n]) || (ar[i][n] == ar[k][n] &&
basis[i] < basis[k])) k = i;
            }
            if (ar[k][n] >= -EPS) break;

            for (j = 0, l = 0; j < n; j++){
                if ((ar[k][j] < (ar[k][l] - EPS)) || (ar[k][j] < (ar[k][l]
- EPS) && index[i] < index[j])){
                    l = j;
                }
            }
            if (ar[k][l] >= -EPS){
                solution_flag = INFEASIBLE; /// No solution is possible
                return -1.0;
            }
            pivot(m, n, k, l);
        }

        for (; ;){
            for (j = 0, l = 0; j < n; j++){
                if ((ar[0][j] < ar[0][l]) || (ar[0][j] == ar[0][l] &&
index[j] < index[l])) l = j;
            }
            if (ar[0][l] > -EPS) break;

            for (i = 1, k = 0; i <= m; i++){
                if (ar[i][l] > EPS && (!k || ar[i][n] / ar[i][l] <
ar[k][n] / ar[k][l] - EPS || (ar[i][n] / ar[i][l] < ar[k][n] / ar[k][l] +
EPS && basis[i] < basis[k]))){
                    k = i;
                }
            }
            if (ar[k][l] <= EPS){
                solution_flag = UNBOUNDED; /// Solution is infinity, no
finite solution exists
                return -666.0;
            }
            pivot(m, n, k, l);
        }

        for (i = 1; i <= m; i++){
            if (basis[i] >= 0) val[basis[i]] = ar[i][n];
        }
        solution_flag = 1; /// Successful completion
        return (ar[0][n] * minmax_flag); /// Negate the output for
MINIMIZE since the objective function was negated
    }
}

int main(){
}
```

```
Simplex OP.cpp
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define EPS 1e-9
#define MAXV 4010
#define MAXC 4010

#define EQUAL 0
#define LESSEQ -1
#define GREATEQ 1
#define MINIMIZE -1
#define MAXIMIZE +1

#define FEASIBLE +1
#define INFEASIBLE -1
#define UNBOUNDED 666

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/***
1.  Simplex Algorithm for Linear Programming
2.  Maximize or minimize f1*x1 + f2*x2 + f3*x3 + ... + fn*xn subject to
some constraints
3.  Constraints are of the form, c1x1 + c2x2 + c3x3 + ... + cnxn (<= or
>= or =) rhs
4.  m is the number of constraints indexed from 1 to m, and n is the
number of variables indexed from 1 to n
5.  It is assumed that all variables satisfies non-negativity constraint,
i.e, xi >= 0
6.  If non-negativity constraint is not desired for a variable x, replace
each occurrence
   by difference of two new variables r1 and r2 (where r1 >= 0 and r2 >=
0, handled automatically by simplex).
   That is, replace every x by r1 - r2 (Number of variables increases by
one, -x, +r1, +r2)
7.  solution_flag = INFEASIBLE if no solution is possible and UNBOUNDED
if no finite solution is possible
8.  Returns the maximum/minimum value of the linear equation satisfying
all constraints otherwise
9. After successful completion, val[] contains the values of x1, x2 ....
xn for the optimal value returned

*** If ABS(X) <= M in constraints, Replace with X <= M and -X <= M

*** Fractional LP:

   max/min
```

```
        3x1 + 2x2 + 4x3 + 6
        -------------------
        3x1 + 3x2 + 2x3 + 5

        s.t. 2x1 + 3x2 + 5x3  0

    Replace with:

    max/min
        3y1 + 2y2 + 4y3 + 6t

        s.t. 3y1 + 3y2 + 2y3 + 5t = 1
        2y1 + 3y2 + 53 - 23t  0

***/

namespace lp{
   double ar[MAXC][MAXV], val[MAXV], rhs[MAXC];
   int n, m, flag, adj[MAXV], idx[MAXV], down[MAXV], link[MAXC];

   void init(int nvar, double func[], int min_or_max){ /// func[] =
objective function
       m = 0, n = nvar, flag = min_or_max;
       for (int i = 1; i <= n; i++) idx[i] = 0;
       for (int i = 1; i <= n; i++) ar[0][i] = func[i] * flag;
   }

   /// var[] = co-efficients of the constraints (LHS), lim = limit in RHS
   /// flag = EQUAL for var[] = lim, LESSEQ for var[] <= lim, GREATEQ for
var[] >= lim
   inline void add_constraint(double var[], double lim, int flag){
       flag *= -1;
       if (flag == 0){
           rhs[++m] = lim;
           for (int i = 1; i <= n; i++) ar[m][i] = var[i];
           rhs[++m] = -lim;
           for (int i = 1; i <= n; i++) ar[m][i] = -var[i];
       }
       else{
           rhs[++m] = lim * flag;
           for (int i = 1; i <= n; i++) ar[m][i] = var[i] * flag;
       }
   }

   void pivot(int x, int y, double& res){ /// pivoting and exchanging a
non-basic variable with a basic variable
            int i, j, len = 0;
       double v = ar[x][y];

       swap(link[x], down[y]);
       rhs[x] /= v, ar[x][y] = 1;
            for (j = 1; j <= n; j++){
           ar[x][j] /= v;
           if (abs(ar[x][j]) > EPS) adj[len++] = j;
```

```
            }

            for (i = 1; i <= m; i++){
            if (abs(ar[i][y]) > EPS && i != x){
                rhs[i] -= ar[i][y] * rhs[x], v = ar[i][y], ar[i][y] = 0;
                for (j = 0; j < len; j++) ar[i][adj[j]] -= (v *
ar[x][adj[j]]);
            }
            }

            res += (ar[0][y] * rhs[x]), v = ar[0][y], ar[0][y] = 0;
            for (j = 0; j < len; j++) ar[0][adj[j]] -= (v *
ar[x][adj[j]]);
    }

    int solve(double& res){ /// simplex core
        int i, j, x, y;
        double u, v, mn, mx;
        for (i = 1; i <= n; i++) down[i] = i;
        for (i = 1; i <= m; i++) link[i] = i + n;

        while (1){ /// phase 1
            x = 0, y = 0, mn = -EPS;
            for (i = 1; i <= m; i++){
                if (rhs[i] < mn) mn = rhs[i], x = i;
            }
            if (x == 0) break;

            for (i = 1; i <= n; i++){
                if (ar[x][i] < -EPS){
                    y = i;
                    if (rand() & 1) break;
                }
            }
            if (y == 0) return INFEASIBLE;
            pivot(x, y, res);
        }

        while (1){ /// phase 2
            x = 0, y = 0, mx = EPS;
            for (i = 1; i <= n; i++){
                if (ar[0][i] > mx) mx = ar[0][i], y = i;
            }
            if (y == 0) break;

            for (i = 1; i <= m; i++){
                if (ar[i][y] > EPS){
                    u = rhs[i] / ar[i][y];
                    if (x == 0 || u < v) x = i, v = u;
                }
            }
            if (x == 0) return UNBOUNDED;
            pivot(x, y, res);
        }
```

```
        res *= flag;
        for (int i = 1; i <= m; i++){
            if(link[i] <= n) idx[link[i]] = i;
        }

        for (int i = 1; i <= n; i++) val[i] = rhs[idx[i]];
            return FEASIBLE;
      }
}

int main(){

}
```

Simplex.cpp

```
#include <bits/stdtr1c++.h>

#define MAXC 1010
#define MAXV 1010
#define EPS 1e-13

#define MINIMIZE -1
#define MAXIMIZE +1
#define LESSEQ -1
#define EQUAL 0
#define GREATEQ 1
#define INFEASIBLE -1
#define UNBOUNDED 666

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/***
1.  Simplex Algorithm for Linear Programming
2.  Maximize or minimize f0*x0 + f1*x1 + f2*x2 + ... + fn-1*xn-1 subject
to some constraints
3.  Constraints are of the form, c0x0 + c1x1 + c2x2 + ... + cn-1xn-1 (<=
or >= or =) lim
4.  m is the number of constraints indexed from 1 to m, and n is the
number of variables indexed from 0 to n-1
5.  ar[0] contains the objective function f, and ar[1] to ar[m] contains
the constraints, ar[i][n] = lim_i
6.  It is assumed that all variables satisfies non-negativity constraint,
i.e, xi >= 0
7.  If non-negativity constraint is not desired for a variable x, replace
each occurrence
   by difference of two new variables r1 and r2 (where r1 >= 0 and r2 >=
0, handled automatically by simplex).
```

```
   That is, replace every x by r1 - r2 (Number of variables increases by one, -x, +r1, +r2)
8.  solution_flag = INFEASIBLE if no solution is possible and UNBOUNDED if no finite solution is possible
9.  Returns the maximum/minimum value of the linear equation satisfying all constraints otherwise
10. After successful completion, val[] contains the values of x0, x1 .... xn for the optimal value returned

*** If ABS(X) <= M in constraints, Replace with X <= M and -X <= M

*** Fractional LP:

   max/min
       3x1 + 2x2 + 4x3 + 6
       -------------------
       3x1 + 3x2 + 2x3 + 5

       s.t. 2x1 + 3x2 + 5x3  0

   Replace with:

   max/min
       3y1 + 2y2 + 4y3 + 6t

       s.t. 3y1 + 3y2 + 2y3 + 5t = 1
       2y1 + 3y2 + 53 - 23t  0

***/

namespace lp{
   long double val[MAXV], ar[MAXC][MAXV];
   int m, n, solution_flag, minmax_flag, basis[MAXC], index[MAXV];

   /// nvars = number of variables, f = objective function, flag = MINIMIZE or MAXIMIZE
   inline void init(int nvars, long double f[], int flag){
       solution_flag = 0;
       ar[0][nvars] = 0.0;
       m = 0, n = nvars, minmax_flag = flag;
       for (int i = 0; i < n; i++){
           ar[0][i] = f[i] * minmax_flag; /// Negating sign of objective function when minimizing
       }
   }

   /// C[] = co-efficients of the constraints (LHS), lim = limit in RHS
   /// cmp = EQUAL for C[] = lim, LESSEQ for C[] <= lim, GREATEQ for C[] >= lim
   inline void add_constraint(long double C[], long double lim, int cmp){
       m++, cmp *= -1;
       if (cmp == 0){
           for (int i = 0; i < n; i++) ar[m][i] = C[i];
           ar[m++][n] = lim;
```

```
            for (int i = 0; i < n; i++) ar[m][i] = -C[i];
            ar[m][n] = -lim;
        }
        else{
            for (int i = 0; i < n; i++) ar[m][i] = C[i] * cmp;
            ar[m][n] = lim * cmp;
        }
    }

    inline void init(){ /// Initialization
        for (int i = 0; i <= m; i++) basis[i] = -i;
        for (int j = 0; j <= n; j++){
            ar[0][j] = -ar[0][j], index[j] = j, val[j] = 0;
        }
    }

    inline void pivot(int m, int n, int a, int b){ /// Pivoting and
exchanging a non-basic variable with a basic variable
        for (int i = 0; i <= m; i++){
            if (i != a){
                for (int j = 0; j <= n; j++){
                    if (j != b){
                        ar[i][j] -= (ar[i][b] * ar[a][j]) / ar[a][b];
                    }
                }
            }
        }

        for (int j = 0; j <= n; j++){
            if (j != b) ar[a][j] /= ar[a][b];
        }
        for (int i = 0; i <= m; i++){
            if (i != a) ar[i][b] = -ar[i][b] / ar[a][b];
        }

        ar[a][b] = 1.0 / ar[a][b];
        swap(basis[a], index[b]);
    }

    inline long double solve(){ /// simplex core
        init();
        int i, j, k, l;
        for (; ;){
            for (i = 1, k = 1; i <= m; i++){
                if ((ar[i][n] < ar[k][n]) || (ar[i][n] == ar[k][n] &&
basis[i] < basis[k] && (rand() & 1))) k = i;
            }
            if (ar[k][n] >= -EPS) break;

            for (j = 0, l = 0; j < n; j++){
                if ((ar[k][j] < (ar[k][l] - EPS)) || (ar[k][j] < (ar[k][l]
- EPS) && index[i] < index[j] && (rand() & 1))){
                    l = j;
                }
```

```
            }
            if (ar[k][l] >= -EPS){
                solution_flag = INFEASIBLE; /// No solution is possible
                return -1.0;
            }
            pivot(m, n, k, l);
        }

        for (; ;){
            for (j = 0, l = 0; j < n; j++){
                if ((ar[0][j] < ar[0][l]) || (ar[0][j] == ar[0][l] &&
index[j] < index[l] && (rand() & 1))) l = j;
            }
            if (ar[0][l] > -EPS) break;

            for (i = 1, k = 0; i <= m; i++){
                if (ar[i][l] > EPS && (!k || ar[i][n] / ar[i][l] <
ar[k][n] / ar[k][l] - EPS || (ar[i][n] / ar[i][l] < ar[k][n] / ar[k][l] +
EPS && basis[i] < basis[k]))){
                    k = i;
                }
            }
            if (ar[k][l] <= EPS){
                solution_flag = UNBOUNDED; /// Solution is infinity, no
finite solution exists
                return -666.0;
            }
            pivot(m, n, k, l);
        }

        for (i = 1; i <= m; i++){
            if (basis[i] >= 0) val[basis[i]] = ar[i][n];
        }
        solution_flag = 1; /// Successful completion
        return (ar[0][n] * minmax_flag); /// Negate the output for
MINIMIZE since the objective function was negated
    }
}

int main(){
}
```

Sparse Table OP.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Sparse Table in O(n), for min and max
```

```
namespace st{
   #define MAX     100010 /// Maximum number of elements in the array
   #define LOG         17 /// (int)floor(log2(MAX)) + 1,  {...7 = 3, 8 = 
4, 9 = 4...}
   #define MAXB      5925 /// (MAX + LG - 1) / LG, (MAX / LG) + 77 works 
fine
   #define op(a, b) ((st::ar[(a)]) < (st::ar[(b)]) ? (a) : (b)) /// 
comparator of two numbers, set to min

   int n, lg, len, T[MAX], ar[MAX], mask[1 << LOG], dp[LOG][MAXB];

   /// Build the sparse table in O(n)
   void init(int m, int* val){
       int i, j, k, d, top;
       for (n = 0; n < m; n++) ar[n] = val[n];

       lg = 32 - __builtin_clz(n);
       d = -1, i = 0, len = (n + lg - 1) / lg;

       while (i < n){
           dp[0][++d] = i++;
           for (j = 1; j < lg && i < n; i++, j++){
               dp[0][d] = op(i, dp[0][d]);
           }
       }

       for (j = 1; j < lg; j++){
           d = (1 << j) >> 1;
           for (i = 0; i < len; i++){
               dp[j][i] = op(dp[j - 1][i], dp[j - 1][i + ((i + d) < len ? 
d : 0)]);
           }
       }

       for (i = 0; i < len; i++){
           top = 0, d = (i * lg) + lg;
           for (j = d - lg; j < n && j < d; j++){
               while (top && ar[j] < ar[mask[top]]) top--; /// Change to 
ar[j] > ar[mask[top]] for max
               T[j] = 1 << (d - j - 1);
               if (top) T[j] |= T[mask[top]];
               mask[++top] = j;
           }
       }

       for (i = 1, k = 1 << lg, d = lg - 1; i < k; i++){
           if (i >= (1 << (lg - d))) d--;
           mask[i] = d;
       }
   }

   /// returns the minimum value (as op is set to min) in the range l-r, 
l must not be greater than r
   inline int query(int l, int r){
```

```
        int c, d, x = (l / lg) + 1, y = (r / lg) - 1, res = l;

        if(x <= y){
            d = lg - mask[y - x + 1] - 1;
            res = op(res, op(dp[d][x], dp[d][y - (1 << d) + 1]));
        }

        c = x * lg, d = y * lg;
        res = op(res, mask[T[(c - 1) < r ? (c - 1) : r] & (~(((1 << (l - c
+ lg )) - 1) << (c - l)))] + c - lg);
        l = l > (d + lg) ? l : d + lg;
        res = op(res, mask[T[r] & (~(((1 << (l - d - lg)) - 1) << (d + (lg
<< 1) - l)))] + d + lg);
        return ar[res]; /// return res to return index of the minimum
value(as op is set to min)
    }
}

int main(){
    return 0;
}
```

Sparse Table.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define LOG 18
#define MAX 100010
#define min(a,b) ((a)<(b) ? (a):(b))
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int n, q, lg[MAX], ar[MAX], dp[LOG][MAX];

int query(int i, int j){
    int k = lg[j - i];
    int x = dp[k][i], y = dp[k][j - (1 << k) + 1];
    return min(x, y);
}

void build(){
    int i, j, l, d, len;
    for (i = 2, lg[0] = lg[1] = 0; i < MAX; i++) lg[i] = lg[i >> 1] + 1;

    for (l = 0; (1 << l) <= n; l++){
        len = 1 << l, d = len >> 1;
        for (i = 0; (i + len) <= n; i++){
            if (!l) dp[l][i] = ar[i];
            else dp[l][i] = min(dp[l - 1][i], dp[l - 1][i + d]);
        }
    }
}
```

```
int main(){
   build();
}
```

Sparse Table.cpp

```
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

namespace st{
   #define LOG  18
   #define MAXN 100010

   int n, lim, lg[MAXN], dp[LOG][MAXN];

   int query(int i, int j){
       int len = lg[j - i];
       return max( dp[len][i], dp[len][j - (1 << len) + 1] );
   }

   void build(int m, int* ar){
       n = m;
       int i, j, l, d, len;

       lg[0] = lg[1] = 0;
       lim = 32 - __builtin_clz(n);

       if (!lg[2]){
           for (i = 2; i < MAXN; i++) lg[i] = lg[i >> 1] + 1;
       }

       for (i = 0; i < n; i++) dp[0][i] = ar[i];
       for (l = 1; l < lim; l++){
           len = (1 << l);
           for (i = 0; (i + len) <= n; i++){
               d = 1 << (l - 1);
               dp[l][i] = max(dp[l - 1][i], dp[l - 1][i + d]);
           }
       }
   }
}

int main(){
}
```

Square Root Decomposition.c

```
#include <stdio.h>
```

```
#include <string.h>
#include <stdbool.h>

#define LIM 50
#define MAX 10010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

bool islucky[MAX], start[100010];
int n, m, d, q, len = 0, lucky[100], ar[100010], pos[100010], add[LIM],
counter[LIM][MAX];

int compare(const void* a, const void* b){
   return (*(int*)a - *(int*)b);
}

void backtrack(int i){
   if (i >= MAX) return;
   if (i){
       islucky[i] = true;
       lucky[len++] = i;
   }
   backtrack((i * 10) + 4);
   backtrack((i * 10) + 7);
}

void Generate(){
   int i, j, k, l;
   m = sqrt(0.5 + (n * len * 1.5));

   clr(add), clr(counter);
   for (i = 0, d = 0; i < n; i += m){
       start[i] = true;
       for (j = 0, k = i; j < m && k < n; j++, k++){
           pos[k] = d;
           counter[d][ar[k]]++;
       }
       d++;
   }
}

void update(int l, int r, int v){
   int i = l;
   while (i <= r) {
       int k = pos[i];
       if (start[i] && (i + m - 1) <= r){
           add[k] += v;
           i += m;
       }
       else{
           counter[k][ar[i]]--;
           ar[i] += v;
           counter[k][ar[i]]++;
           i++;
```

```
        }
    }
}

int count(int l, int r){
    int i = l, j = 0, res = 0;
    while (i <= r) {
        int k = pos[i];
        if (start[i] && (i + m - 1) <= r){
            for (j = len - 1; j >= 0; j--){
                int x = lucky[j] - add[k];
                if (x < 0) break;
                res += counter[k][x];
            }
            i += m;
        }
        else res += islucky[ar[i++] + add[k]];
    }
    return res;
}

int main(){
    backtrack(0);
    qsort(lucky, len, sizeof(int), compare);

    char str[15];
    int i, j, k, l, r, d, v;
    while (scanf("%d %d", &n, &q) != EOF){
        for (i = 0; i < n; i++) scanf("%d", &ar[i]);
        Generate();

        while (q--){
            scanf("%s %d %d", str, &l, &r);
            if (str[0] == 'c') printf("%d\n", count(--l, --r));
            else{
                scanf("%d", &v);
                update(--l, --r, v);
            }
        }
    }
    return 0;
}
```

String Hash + Segment Tree.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;
```

```
/// String Hash with Segment Tree
/// Allows setting range with a character, querying hash of a range in
O(log n)
/// String functions uses 0-based index, however segment tree uses 1-
based index

namespace strhash{
   #define LET 10 /// Number of distinct letters in string, Set to digits
[0-9] by default
   #define ADD 10007 /// Just another prime added randomly to get better
hash values :)
   #define MAX 100010

   int P[2][MAX], RD[2][LET][MAX];
   int n, ar[MAX], lazy[MAX * 4];
   const int MOD[] = {2078526727, 2117566807};
   const int BASE[] = {1572872831, 1971536491};

   void build_tree(int idx, int a, int b);

   inline int getval(char ch){ /// Value of a character (without adding
ADD)
       return ch - 48; /// Change this for lowercase or uppercase letters
   }

   void precalc(){ /// Call precalc() just once in whole program
       int i, j, k, d;
       P[0][0] = P[1][0] = 1;
       for (i = 1; i < MAX; i++){
           P[0][i] = ((long long) P[0][i - 1] * BASE[0]) % MOD[0];
           P[1][i] = ((long long) P[1][i - 1] * BASE[1]) % MOD[1];
       }

       for (i = 0; i < 2; i++){
           for (d = 0; d < LET; d++){
               k = ADD + d;
               long long x = 0;
               for (j = 1; j <= MAX; j++){
                   x = ((x * BASE[i]) + k) % MOD[i];
                   RD[i][d][j] = x;
               }
           }
       }
   }

   void init(char* str){
       n = strlen(str);
       for (int i = 0; i < n; i++) ar[i + 1] = getval(str[i]) + ADD;
       build_tree(1, 1, n);
   }

   struct Node{
       int H[2];
```

```
        inline Node(){
        }

        inline Node(int h1, int h2){
            H[0] = h1, H[1] = h2;
        }
    } tree[MAX << 2];

    inline void propagate(int idx, int a, int b){
        if (lazy[idx]){
            int p = (idx << 1), q = p + 1;
            int c = (a + b) >> 1, d = c + 1, val = lazy[idx];

            if (a != b){
                lazy[p] = lazy[q] = lazy[idx];
                tree[p] = Node(RD[0][val - ADD][c - a + 1], RD[1][val -
ADD][c - a + 1]);
                tree[q] = Node(RD[0][val - ADD][b - d + 1], RD[1][val -
ADD][b - d + 1]);
            }
            lazy[idx] = 0;
        }
    }

    inline void build_tree(int idx, int a, int b){
        lazy[idx] = 0;
        if (a == b){
            tree[idx].H[0] = tree[idx].H[1] = ar[a];
            return;
        }

        int p = (idx << 1), q = p + 1;
        int c = (a + b) >> 1, d = c + 1;

        build_tree(p, a, c);
        build_tree(q, d, b);

        tree[idx].H[0] = ((tree[p].H[0] * (long long) P[0][b - d + 1]) +
tree[q].H[0]) % MOD[0];
        tree[idx].H[1] = ((tree[p].H[1] * (long long) P[1][b - d + 1]) +
tree[q].H[1]) % MOD[1];
    }

    inline void update(int idx, int a, int b, int l, int r, int val){
        if (a == l && b == r){
            lazy[idx] = val;
            tree[idx] = Node(RD[0][val - ADD][b - a + 1], RD[1][val -
ADD][b - a + 1]);
            propagate(idx, a, b);
            return;
        }

        propagate(idx, a, b);
        int p = (idx << 1), q = p + 1;
```

```
        int c = (a + b) >> 1, d = c + 1;

        if (r <= c) update(p, a, c, l, r, val);
        else if (l >= d) update(q, d, b, l, r, val);
        else{
            update(p, a, c, l, c, val);
            update(q, d, b, d, r, val);
        }

        tree[idx].H[0] = ((tree[p].H[0] * (long long) P[0][b - d + 1]) +
tree[q].H[0]) % MOD[0];
        tree[idx].H[1] = ((tree[p].H[1] * (long long) P[1][b - d + 1]) +
tree[q].H[1]) % MOD[1];
    }

    inline Node query(int idx, int a, int b, int l, int r){
        propagate(idx, a, b);
        int p = (idx << 1), q = p + 1;
        int c = (a + b) >> 1, d = c + 1;

        if (a == l && b == r) return tree[idx];
        if (r <= c) return query(p, a, c, l, r);
        else if (l >= d) return query(q, d, b, l, r);
        else{
            Node x = query(p, a, c, l, c);
            Node y = query(q, d, b, d, r);
            for (int i = 0; i < 2; i++){
                x.H[i] = ((P[i][r - d + 1] * (long long) x.H[i]) + y.H[i])
% MOD[i];
            }
            return x;
        }
    }

    /// Note 0-based index used for string update and gethash
    inline void update(int l, int r, char ch){ /// Sets the sub-string in
str[l:r] to ch
        update(1, 1, n, ++l, ++r, getval(ch) + ADD);
    }

    inline long long gethash(int l, int r){
        Node h = query(1, 1, n, ++l, ++r);
        return (h.H[0] << 32) | h.H[1];
    }
}

int main(){

}
```

String Hash.c

```
---------------------------------------------------
#include <stdio.h>
#include <string.h>
```

```
#include <stdbool.h>

#define MAX 1000010
#define min(a,b) ((a)<(b)?(a):(b))
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

const unsigned long long base = 1925728309;

int n;
char str[MAX];
unsigned long long ar[3][MAX];

void Generate(char* str, int n, unsigned long long ar[3][MAX]){
   int i;
   unsigned long long x;

   ar[0][0] = 1;
   for (i = 1; i <= n; i++) ar[0][i] = (ar[0][i - 1] * base);

   x = 0;
   for (i = 0; i < n; i++){
       x = (x * base) + str[i];
       ar[1][i] = x;
   }

   x = 0;
   for (i = n - 1; i >= 0; i--){
       x = (x * base) + str[i];
       ar[2][i] = x;
   }
}

unsigned long long forward_hash(int i, int j, unsigned long long
ar[3][MAX], int n){
   unsigned long long x = ar[1][j];
   if (i){
       unsigned long long y = ar[0][j - i + 1] * ar[1][i - 1];
       x -= y;
   }
   return x;
}

unsigned long long reverse_hash(int i, int j, unsigned long long
ar[3][MAX], int n){
   unsigned long long x = ar[2][i];
   if ((j + 1) != n){
       unsigned long long y = ar[0][j - i + 1] * ar[2][j + 1];
       x -= y;
   }
   return x;
}

int main(){
```

```
}
```

String Hash.cpp

```
#include <bits/stdtr1c++.h>

#define MAXLEN 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

class StringHash{
   public:

   int n;
   char str[MAXLEN];
   unsigned long long base = 1925728309;
   unsigned long long P[MAXLEN], ar[MAXLEN], rev[MAXLEN];

   StringHash(){
   }

   StringHash(char* temp){
       strcpy(str, temp);
       n = strlen(str);
       Generate();
   }

   void Generate(){
       int i;
       unsigned long long x;

       P[0] = 1;
       for (i = 1; i <= n; i++) P[i] = (P[i - 1] * base);

       x = 0;
       for (i = 0; i < n; i++){
           x = (x * base) + str[i];
           ar[i] = x;
       }

       x = 0;
       for (i = n - 1; i >= 0; i--){
           x = (x * base) + str[i];
           rev[i] = x;
       }
   }

   unsigned long long forward_hash(int i, int j){
       unsigned long long x = ar[j];
       if (i){
           unsigned long long y = P[j - i + 1] * ar[i - 1];
```

```
            x -= y;
        }
        return x;
    }

    unsigned long long reverse_hash(int i, int j){
        unsigned long long x = rev[i];
        if ((j + 1) != n){
            unsigned long long y = P[j - i + 1] * rev[j + 1];
            x -= y;
        }
        return x;
    }
};

int main(){
    char* str = "Hello World";
    StringHash S = StringHash(str);
}
```

Subtree To Array.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 100010

using namespace std;

/// 1 based index for arrays and trees
vector <int> adj[MAX];
int r, S[MAX], E[MAX]; /// S[i] = starting index of subtree rooted at i,
E[i] = ending index of the subtree rooted at i

void dfs(int i, int p){
    if (n == 1) S[i] = E[i] = ++r;
    int j, x, len = adj[i].size();

    S[i] = ++r;
    for (j = 0; j < len; j++){
        if (adj[i][j] != p) dfs(adj[i][j], i);
    }
    E[i] = r;
}

int main(){
    r = 0;
    dfs(1, 0); /// 1 is root
}
```

Sudoku Solver.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```c
#include <stdbool.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

struct Sudoku{
   int i, j, x;
};

bool flag;
char str[10];
struct Sudoku adj[81];
int A[9], B[9], C[3][3];
int t, len, steps, n = 9, T[1 << 10], dv[10], row[9], col[9], box[3][3],
ar[9][9];

int compare(const void* a, const void* b){
   return ((*(struct Sudoku*)b).x - (*(struct Sudoku*)a).x);
}

int F(){
   int i, j, k, l, d, x;

   clr(A), clr(B), clr(C);
   for (i = 0; i < n; i++){
       for (j = 0; j < n; j++){
           x = ar[i][j];
           if (x){
               x--;
               A[i] |= (1 << x);
               B[j] |= (1 << x);
               C[i / 3][j / 3] |= (1 << x);
           }
       }
   }

   int idx = 0, r = -1;
   for (d = 0; d < len; d++){
       i = adj[d]. i, j = adj[d].j;
       if (!ar[i][j]){
           k = __builtin_popcount(A[i] | B[j] | box[i / 3][j / 3]);
           if (k > r) r = k, idx = d;
           if (r == n) return -1;
       }
   }

   return idx;
}

void backtrack(int idx){
   steps++;
   if (idx == len){
       int k, l;
       flag = true;
```

```
        for (k = 0; k < n; k++){
            for (l = 0; l < n; l++){
                printf("%d ", ar[k][l]);
            }
            puts("");
        }
        return;
    }

    int lol = F();
    if (lol == -1) return;
    int y, d, i = adj[lol].i, j = adj[lol].j, k = dv[i], l = dv[j];
    int x = ~(row[i] | col[j] | box[k][l]) & 511;

    while (x){
        y = (-x & x);
        d = T[y];

        ar[i][j] = d + 1, row[i] |= (1 << d), col[j] |= (1 << d),
box[k][l] |= (1 << d);
        backtrack(idx + 1);
        ar[i][j] = 0, row[i] &= ~(1 << d), col[j] &= ~(1 << d), box[k][l]
&= ~(1 << d);

        if (flag) return;
        x ^= y;
    }
}

int main(){
    int i, j, k, l, x;
    for (i = 0; i < 10; i++) T[1 << i] = i, dv[i] = (i / 3);

    scanf("%d", &t);
    while (t--) {
        clr(ar), clr(row), clr(col), clr(box);

        for (i = 0; i < n; i++){
            for (j = 0; j < n; j++){
                scanf("%d", &ar[i][j]);
                if (!ar[i][j]) continue;

                x = ar[i][j] - 1;
                row[i] |= (1 << x);
                col[j] |= (1 << x);
                box[i / 3][j / 3] |= (1 << x);
            }
        }

        len = 0;
        for (i = n - 1; i >= 0; i--){
            for (j = n - 1; j >= 0; j--){
                if (!ar[i][j]){
                    x = row[i] | col[j] | box[i / 3][j / 3];
```

```
                    adj[len].i = i, adj[len].j = j;
                    adj[len++].x = __builtin_popcount(x);
                }
            }
        }
        qsort(adj, len, sizeof(struct Sudoku), compare);

        flag = false, steps = 0;
        backtrack(0);
        if (!flag) puts("No solution");
    }
    return 0;
}
```

Suffix Array.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 100010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

char str[MAX];
int s0[(MAX / 3) + 10], sa0[(MAX / 3) + 10];
int n, ar[MAX], sa[MAX], lcp[MAX], bucket[MAX], mem[MAX << 2];

void radixsort(int* source, int* dest, int* val, int n, int lim){ ///
hash = 349247
    int i, s = 0, x;
    memset(bucket, 0, lim << 2);
    for (i = 0; i < n; i++) bucket[val[source[i]]]++;

    for (i = 0; i < lim; i++){
        x = bucket[i];
        bucket[i] = s, s += x;
    }
    for (i = 0; i < n; i++) dest[bucket[val[source[i]]]++] = source[i];
}

void DC3(int* ar, int* sa, int n, int lim, int ptr){ /// hash = 758459
    int *s12, *sa12;
    int allc = (n / 3) << 1, n0 = (n + 2) / 3;
    int i, j, k, l, c, d, p, t, m, r, counter;
    s12 = &mem[ptr], ptr += (allc + 5), sa12 = &mem[ptr], ptr += (allc +
5);

    c = 0, m = 0, r = n + ((n % 3) == 1);
    for (i = 0; i < r; i++, m++){
        if (m == 3) m = 0;
        if (m) s12[c++] = i;
    }
```

```
    s12[c] = sa12[c] = s12[c + 1] = sa12[c + 1] = s12[c + 2] = sa12[c + 2]
= 0;
    radixsort(s12, sa12, ar + 2, c, lim + 1);
    radixsort(sa12, s12, ar + 1, c, lim + 1);
    radixsort(s12, sa12, ar, c, lim + 1);

    counter = 0, j = -1;
    for (i = 0; i < c; i++){
        if ((ar[sa12[i]] != j) || (ar[sa12[i] + 1] != k) || (ar[sa12[i] +
2] != l)){
            counter++;
            j = ar[sa12[i]], k = ar[sa12[i] + 1], l = ar[sa12[i] + 2];
        }
        if((sa12[i] % 3) == 1) s12[sa12[i] / 3] = counter;
        else s12[(sa12[i] / 3) + n0] = counter;
    }

    if (counter == c){
        for(i = 0; i < c; i++) sa12[s12[i] - 1] = i;
    }
    else{
        DC3(s12, sa12, c, counter, ptr);
        for (i = 0; i < c; i++) s12[sa12[i]] = i + 1;
    }

    for (i = 0, d = 0; i < c; i++){
        if (sa12[i] < n0) s0[d++] = (sa12[i] * 3);
    }
    radixsort(s0, sa0, ar, d, lim + 1);
    for (k = 0, l = ((n % 3) == 1), r = 0; r < n; r++){
        j = sa0[k];
        i = ((sa12[l] < n0) ? (sa12[l] * 3) + 1 : ((sa12[l] - n0) * 3) +
2);
        if (l == c) sa[r] = sa0[k++];
        else if (k == d) sa[r] = i, l++;
        else{
            if (sa12[l] < n0){
                if ((ar[i] < ar[j]) || (ar[i] == ar[j] && s12[sa12[l] +
n0] <= s12[j / 3])) sa[r] = i, l++;
                else sa[r] = j, k++;
            }
            else{
                if ((ar[i] < ar[j]) || (ar[i] == ar[j] && ar[i + 1] < ar[j
+ 1]) || (ar[i] == ar[j] && ar[i + 1] == ar[j + 1] && s12[sa12[l] - n0 +
1] <= s12[(j /3) + n0]) ) sa[r] = i, l++;
                else sa[r] = j, k++;
            }
        }
    }
}

void LcpArray(){ /// hash = 935019
    int i, j, k;
    for (i = 0; i < n; i++) ar[sa[i]] = i;
```

```
    for (k = 0, i = 0; i < n; i++, k?k--:0){
        if (ar[i] == (n - 1)) k = 0;
        else{
            j = sa[ar[i] + 1];
            while(((i + k) < n) && ((j + k) < n) && (str[i + k] == str[j +
k])) k++;
        }
        lcp[ar[i]] = k;
    }
}

void Generate(){
    int i, j, lim = 0;
    for (i = 0; i < n; i++){
        ar[i] = str[i];
        if (ar[i] > lim) lim = ar[i];
    }

    ar[n] = ar[n + 1] = ar[n + 2] = 0;
    DC3(ar, sa, n, lim, 0);
}

int main(){
    scanf("%s", str);
    n = strlen(str);
    Generate();
    LcpArray();
    return 0;
}
```

Suffix Array.cpp

```
#include <bits/stdtr1c++.h>

#define LOG 18
#define MAX 50010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

namespace sa{
    char str[MAX];
    int n, m, freq[MAX], ar[MAX], lcp[MAX], temp[MAX], dp[LOG][MAX];

    int getlcp(int a, int b) {
        int i, l, res = 0;
        for (i = LOG - 1; i >= 0; i--) {
            l = 1 << i;
            if (l <= n && dp[i][a] == dp[i][b]) res += l, a += l, b += l;
        }
```

```
        return res;
    }

    long long distinct_substrings(){
        long long res = ((long long)n * (n + 1)) >> 1;
        for (int i = 0; (i + 1) < n; i++) res -= lcp[i];
        return res;
    }

    void build(char* hello, int gen_lcp){
        int i, j, k, l, h;
        strcpy(str, hello);
        n = strlen(str) + 1;

        clr(freq);
        for (i = 0; i < n; i++) freq[str[i]]++;
        for (i = 1; i < 256; i++) freq[i] += freq[i - 1];

        for (i = 0; i < n; i++){
            ar[freq[str[i]] - 1] = i;
            freq[str[i]]--;
        }

        dp[0][ar[0]] = 1;
        for (i = 1, m = 1; i < n; i++) {
            if (str[ar[i]] != str[ar[i - 1]]) m++;
            dp[0][ar[i]] = m;
        }

        for (i = 1; ;i++) {
            h = 1 << (i - 1);
            for (j = 0; j < n; j++) {
                temp[j] = ar[j] - h;
                if (temp[j] < 0) temp[j] += n;
            }

            for (j = 1; j <= m; j++) freq[j] = 0;
            for (j = 0; j < n; j++)  freq[dp[i - 1][temp[j]]]++;
            for (j = 2; j <= m; j++) freq[j] += freq[j - 1];

            for (j = n - 1; j >= 0; j--) {
                ar[freq[dp[i - 1][temp[j]]] - 1] = temp[j];
                freq[dp[i - 1][temp[j]]]--;
            }

            dp[i][ar[0]] = 1;
            for (j = 1, m = 1; j < n; j++) {
                k = ar[j] + h, l = ar[j - 1] + h;
                if (k >= n) k -= n;
                if (l >= n) l -= n;
                if ((dp[i - 1][ar[j]] != dp[i - 1][ar[j - 1]]) || (dp[i -
1][k] != dp[i - 1][l])) m++;
                dp[i][ar[j]] = m;
            }
```

```
            if ((1 << i) >= n) break;
        }

        for (i = 0; i < n; i++) ar[i] = ar[i + 1];
        n--;

        if (gen_lcp){
            for (i = 0; i < n; i++) temp[ar[i]] = i;
            for (k = 0, i = 0; i < n; i++, k ? k-- : 0){
                if (temp[i] == (n - 1)) k = 0;
                else{
                    j = ar[temp[i] + 1];
                    while(((i + k) < n) && ((j + k) < n) && (str[i + k] ==
str[j + k])) k++;
                }
                lcp[temp[i]] = k;
            }
        }
    }
}

int main(){
}


namespace suffix_array{ /// N log^N
    int n, g;
    vector <int> sa, pos, temp;

    inline bool compare(int i, int j){
        if (pos[i] != pos[j]) return (pos[i] < pos[j]);
        i += g, j += g;
        return (i < n && j < n) ? (pos[i] < pos[j]) : (i > j);
    }

    vector<int> construct(char* str){
        n = strlen(str);
        sa.resize(n, 0), pos.resize(n, 0), temp.resize(n, 0);

        for (int i = 0; i < n; i++) sa[i] = i, pos[i] = str[i];
        for (g = 1; ; g<<= 1){
            sort(sa.begin(), sa.end(), compare);
            for (int i = 0; (i + 1) < n; i++) temp[i + 1] = temp[i] +
compare(sa[i], sa[i + 1]);
            for (int i = 0; i < n; i++) pos[sa[i]] = temp[i];
            if (temp[n - 1] == n - 1) break;
        }
        return sa;
    }
};
```

Sum Of Powers - Lagrange Polynomial OP.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 1000010
#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

namespace lgr{
   short factor[MAX];
   int S[MAX], ar[MAX], inv[MAX];

   inline int expo(int a, int b){
       int res = 1;

       while (b){
           if (b & 1) res = (long long)res * a % MOD;
           a = (long long)a * a % MOD;
           b >>= 1;
       }
       return res;
   }

   int lagrange(long long n, int k){
       if (!k) return (n % MOD);

       int i, j, x, y, res = 0;
       if (!inv[0]){
           for (i = 2, x = 1; i < MAX; i++) x = (long long)x * i % MOD;
           inv[MAX - 1] = expo(x, MOD - 2);
           for (i = MAX - 2; i >= 0; i--) inv[i] = ((long long)inv[i + 1]
* (i + 1)) % MOD;

           for (i = 0; i < MAX; i++) factor[i] = 0;
           for (i = 4; i < MAX; i += 2) factor[i] = 2;
           for (i = 3; (i * i) < MAX; i += 2){
               if (!factor[i]){
                   for (j = (i * i), x = i << 1; j < MAX; j += x){
                       factor[j] = i;
                   }
               }
           }
       }

       k++;
       for (ar[1] = 1, ar[0] = 0, i = 2; i <= k; i++){
           if (!factor[i]) ar[i] = expo(i, k - 1);
           else ar[i] = ((long long)ar[factor[i]] * ar[i / factor[i]]) %
MOD;
       }
```

```
        for (i = 1; i <= k; i++){
            ar[i] += ar[i - 1];
            if (ar[i] >= MOD) ar[i] -= MOD;
        }
        if (n <= k) return ar[n];

        for (S[k] = 1, i = k - 1; i >= 0; i--) S[i] = ((long long)S[i + 1]
* ((n - i - 1) % MOD)) % MOD;
        for (i = 0, y = 1; i <= k; i++){
            x = (long long)ar[i] * y % MOD * S[i] % MOD * inv[k - i] % MOD
* inv[i] % MOD;
            if ((k - i) & 1){
                res -= x;
                if (res < 0) res += MOD;
            }
            else{
                res += x;
                if (res >= MOD) res -= MOD;
            }
            y = ((long long)y * ((n - i) % MOD)) % MOD;
        }

        return (res % MOD);
    }
}

int main(){
    int k;
    long long n;
    while (scanf("%lld %d", &n, &k) != EOF){
        printf("%d\n", lgr::lagrange(n, k));
    }
    return 0;
}
```

Sum Of Powers - Lagrange Polynomial.cpp

```
#include <bits/stdtr1c++.h>

#define MAX 1000010
#define MOD 1000000007
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

namespace lgr{ /// hash = 985021
    int F[MAX], P[MAX], S[MAX];

    int expo(int a, int b){
        long long res = 1;

        while (b){
```

```
            if (b & 1) res = res * a % MOD;
            a = (long long)a * a % MOD;
            b >>= 1;
        }
        return res;
    }

    int lagrange(long long n, int k){
        long long res = 0;
        int i, x, y, z, m = k + 2, sum = 0;

        if (!F[0]){
            for (F[0] = 1, i = 1; i < MAX; i++) F[i] = ((long long)F[i -
1] * i) % MOD;
        }

        P[0] = S[m + 1] = 1;
        for (i = 1; i <= m; i++) P[i] = ((long long)P[i - 1] * (((n - i) %
MOD) + MOD)) % MOD;
        for (i = m; i >= 1; i--) S[i] = ((long long)S[i + 1] * (((n - i) %
MOD) + MOD)) % MOD;

        for (i = 1; i <= m; i++){
            sum += expo(i, k);
            if (sum >= MOD) sum -= MOD;
            x = ((long long)P[i - 1] * S[i + 1]) % MOD;
            y = ((long long)F[i - 1] * F[m - i]) % MOD;
            if ((m - i) & 1) y = MOD - y;
            z = ((long long)x * expo(y, MOD - 2)) % MOD;
            res = (res + ((long long)z * sum)) % MOD;
        }

        return (res % MOD);
    }
}

int main(){
    int k;
    long long n;
    cin >> n >> k;
    cout << lgr::lagrange(n, k) << endl;
    return 0;
}
```

Sum of Divisors in Range.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define SQR 10001
#define MAX 100000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
```

```
short P[MAX];
int L[MAX], ar[MAX];

void Generate(){
   int i, j, k, l, d, x, y, z, p;

   P[0] = P[1] = L[0] = L[1] = 1;
   for (i = 4; i < MAX; i++, i++) P[i] = 2;

   for (i = 3; i < SQR; i++, i++){
       if (!P[i]){
           d = i << 1;
           for (j = (i * i); j < MAX; j += d) P[j] = i;
       }
   }

   for (i = 2; i < MAX; i++){
       if (!P[i]) L[i] = i;
       else{
           L[i] = P[i];
           x = L[i /P[i]];
           if (x > L[i]) L[i] = x;
       }
   }

   ar[0] = 0, ar[1] = 1;
   for (i = 2; i < MAX; i++){
       if (L[i] == i) ar[i] = i + 1;
       else{
           x = i, y = 1, p = L[i];
           while (L[x] == L[i]){
               y += p;
               x /= L[i], p *= L[i];
           }
           ar[i] = (ar[x] * y);
       }
   }
}

int main(){
   Generate();
   int t, n, i, r;
   return 0;
}
```

Thomas Algorithm.cpp

```
--------------------------------------------------
#include <stdio.h>
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
```

```
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Equation of the form: (x_prev * l) + (x_cur * p) + (x_next * r) = rhs

struct equation{
   long double l, p, r, rhs;

   equation(){}
   equation(long double l, long double p, long double r, long double rhs
= 0.0):
       l(l), p(p), r(r), rhs(rhs){}
};

/// Thomas algorithm to solve tri-digonal system of equations in O(n)

vector <long double> thomas_algorithm(int n, vector <struct equation>
ar){
   ar[0].r = ar[0].r / ar[0].p;
   ar[0].rhs = ar[0].rhs / ar[0].p;

   for (int i = 1; i < n; i++){
       long double v = 1.0 / (ar[i].p - ar[i].l * ar[i - 1].r);
       ar[i].r = ar[i].r * v;
       ar[i].rhs = (ar[i].rhs - ar[i].l * ar[i - 1].rhs) * v;
   }
   for (int i = n - 2; i >= 0; i--) ar[i].rhs = ar[i].rhs - ar[i].r *
ar[i + 1].rhs;

   vector <long double> res;
   for (int i = 0; i < n; i++) res.push_back(ar[i].rhs);
   return res;
}

int main(){

}
```

Treap (Static).cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAXN 200010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

struct node{
   node *l, *r;
   int key, subtree, priority;
```

```
   inline node(){
       l = r = 0;
   }

   inline node(int val){
       l = r = 0;
       subtree = 1, key = val;
       priority = (rand() << 16) ^ rand();
   }

   inline void update(){
       subtree = 1;
       if (l) subtree += l->subtree;
       if (r) subtree += r->subtree;
   }
} pool[MAXN]; /// Maximum number of nodes in treap

struct Treap{
   int idx; /// Make global if multiple copy of treaps required as of
some moment
   struct node* root;

   inline void merge(node* &cur, node* l, node* r){
       if (!l || !r) cur = l ? l : r;
       else if (l->priority > r->priority) merge(l->r, l->r, r), cur = l;
       else merge(r->l, l, r->l), cur = r;
       if (cur) cur->update();
   }

   /// Splits treap into 2 treaps l and r such that all values in l <=
key and all values in r > key
   inline void split(node* cur, node* &l, node* &r, int key){
       if (!cur) l = r = 0;
       else if (key <= cur->key) split(cur->l, l, cur->l, key), r = cur;
       else split(cur->r, cur->r, r, key), l = cur;
       if (cur) cur->update();
   }

   inline void insert(node* &cur, node* it){
       if (!cur) cur = it;
       else if (it->priority > cur->priority) split(cur, it->l, it->r,
it->key), cur = it;
       else insert((it->key < cur->key)? cur->l : cur->r, it);
       if (cur) cur->update();
   }

   inline void erase(node* &cur, int key){
       if (!cur) return;
       if (cur->key == key) merge(cur, cur->l, cur->r);
       else erase((cur->key > key) ? cur->l : cur->r, key);
       if (cur) cur->update();
   }

   Treap(){
```

```
        srand(time(0));
        idx = 0, root = 0; /// Remove idx = 0 and include in main to reset
all
                         /// if multiple copy of treaps required as of
some moment
    }

    inline void insert(int key){
        pool[idx] = node(key);
        insert(root, &pool[idx++]);
    }

    inline void erase(int key){
        erase(root, key);
    }

    inline int size(){
        if (root) return root->subtree;
        return 0;
    }

    /// Returns the k'th smallest element of the treap in 1-based index, -
1 on failure
    inline int kth(int k){
        if ((k < 1) || (k > size())) return -1;

        node *l, *r, *cur = root;
        for (; ;){
            l = cur->l, r = cur->r;
            if (l){
                if (k <= l->subtree) cur = l;
                else if ((l->subtree + 1) == k) return cur->key;
                else cur = r, k -= (l->subtree + 1);
            }
            else{
                if (k == 1) return (cur->key);
                else cur = r, k--;
            }
        }
    }

    /// Returns the count of keys less than x in the treap
    inline int count(int key){
        int res = 0;
        node *l, *r, *cur = root;

        while (cur){
            l = cur->l, r = cur->r;
            if (cur->key < key) res++;
            if (key < cur->key) cur = l;
            else{
                cur = r;
                if (l) res += l->subtree;
            }
```

```
        }
        return res;
    }
};

int main(){

}
```

Treap (Static, Split And Merge).cpp

```
#include <bits/stdtr1c++.h>

#define MAXN 200010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

int pool_index = 0;

struct node{
   node *l, *r, *parent;
   int key, subtree, priority;

   inline node(){
       l = r = 0, parent = 0;
   }

   inline node(int val){
       l = r = 0, parent = 0;
       subtree = 1, key = val;
       priority = (rand() << 16) ^ rand();
   }

   inline void update(){
       subtree = 1;
       if (l){
           l->parent = this;
           subtree += l->subtree;
       }
       if (r){
           r->parent = this;
           subtree += r->subtree;
       }
   }
} pool[MAXN];

struct Treap{
   struct node* root;

   inline int size(node* &cur){
       if (cur) cur->update();
```

```
        return (cur ? cur->subtree : 0);
    }

    inline int size(){
        return (root ? root->subtree : 0);
    }

    inline void merge(node* &cur, node* l, node* r){
        if (!l || !r) cur = l ? l : r;
        else if (l->priority > r->priority) merge(l->r, l->r, r), cur = l;
        else merge(r->l, l, r->l), cur = r;
        if (cur) cur->update();
    }

    void split(node* cur, node* &l, node* &r, int key) {
        l = 0, r = 0;
        if (!cur) return;

        if (cur->key < key) {
            l = cur;
                  split(l->r, l->r, r, key);
                  l->update();
        }
        else{
            r = cur;
                  split(r->l, l, r->l, key);
                  r->update();
        }
    }

    void split_index(node* cur, node* &l, node* &r, int index) {
        l = 0, r = 0;
        if (!cur) return;

        if (size(cur->l) < index) {
            l = cur;
            split_index(l->r, l->r, r, index - size(cur->l) - 1);
            if (l) l->update();
        }
        else {
            r = cur;
            split_index(r->l, l, r->l, index);
            if (r) r->update();
        }
    }

    Treap(){
        root = 0;
        pool_index = 0; /// Remove if multiple copies of treap required at
the same time
    }

    inline void insert(int key){
        node* l, *r;
```

```
        split(root, l, r, key);
        pool[pool_index] = node(key);
        merge(root, l, &pool[pool_index++]);
        merge(root, root, r);
    }

    inline bool erase(int key){
        node *l, *r, *m;
        split(root, l, r, key);
        split_index(r, m, r, 1);
        bool res = (m && m->key == key);
        if (!res) merge(r, m, r);
        merge(root, l, r);
        return res;
    }

    inline int rank(node* cur){ /// rank of node in the treap
        int res = 1 + size(cur->l);
        while (cur->parent){
            if (cur->parent->r == cur) res += (size(cur->parent->l) + 1);
            cur = cur->parent;
        }
        return res;
    }

    /// Returns the k'th smallest element of treap in 1-based index, -1 on
failure
    inline int kth(int k){
        if ((k < 1) || (k > size())) return -1;

        node *l, *r, *m;
        split_index(root, l, r, k);
        split_index(l, l, m, l->subtree - 1);
        int res = m->key;
        merge(l, l, m);
        merge(root, l, r);
        return res;
    }

    /// Returns the count of keys less than x in treap
    inline int count(int key){
        node *l, *r, *cur = root;
        split(root, l, r, key);
        int res = (l ? l->subtree : 0);
        merge(root, l, r);
        return res;
    }
};

int main(){
    char str[10];
    int n, i, j, x, res;
    tr1::unordered_set <int> S;
```

```
    scanf("%d", &n);
    Treap T = Treap();
    while (n--){
        scanf("%s %d", str, &x);
        if (str[0] == 'I' && !S.count(x)) S.insert(x), T.insert(x);
        if (str[0] == 'D') S.erase(x), T.erase(x);
        if (str[0] == 'C') printf("%d\n", T.count(x));
        if (str[0] == 'K'){
            res = T.kth(x);
            if (res == -1) puts("invalid");
            else printf("%d\n", res);
        }
    }
    return 0;
}
```

Treap With Implicit Keys (Run Length Encoding + Mod Delete).cpp
----

```
#include <stdio.h>
#include <bits/stdtr1c++.h>

#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

struct node{
    node *l, *r;
    unsigned int priority;
    long long a, d, n, minval, maxval, tree_size;

    inline node(){
        l = r = 0;
    }

    inline node(long long first_term, long long common_diff, long long
nterms){
        l = r = 0;
        if (nterms == 1) common_diff = 1;
        priority = (rand() << 16) ^ rand();
        a = first_term, d = common_diff, n = tree_size = nterms;
        minval = a, maxval = a + (n - 1) * d;
    }

    inline void update(){
        assert(n > 0);
        tree_size = n, minval = a, maxval = a + (n - 1) * d;
        if (l){
            tree_size += l->tree_size;
            minval = min(minval, l->minval);
            maxval = max(maxval, l->maxval);
        }
```

```
        if (r){
            tree_size += r->tree_size;
            minval = min(minval, r->minval);
            maxval = max(maxval, r->maxval);
        }
    }
} pool[8000010];

struct Treap{
    int idx;
    struct node* root;

    Treap(long long n){
        idx = 0;
        pool[idx] = node(1, 1, n);
        root = &pool[idx++];
    }

    inline long long size(){
        if (root) return root->tree_size;
        return 0;
    }

    inline long long size(node* &cur){
        if (cur) cur->update();
        return (cur ? cur->tree_size : 0);
    }

    inline void merge(node* &cur, node* l, node* r){
        if (!l || !r) cur = l ? l : r;
        else if (l->priority > r->priority) merge(l->r, l->r, r), cur = l;
        else merge(r->l, l, r->l), cur = r;
        if (cur) cur->update();
    }

    inline void split(node* cur, node* &l, node* &r, long long key, long
long counter = 0){
        if (!cur){
            l = r = 0;
            return;
        }

        long long cur_key = counter + (cur->l ? cur->l->tree_size : 0);
        if (key <= cur_key) split(cur->l, l, cur->l, key, counter), r =
cur;
        else split(cur->r, cur->r, r, key, cur_key + cur->n), l = cur;
        if (cur) cur->update();
    }

    inline void divide(node* &cur, long long i){
        if (i < 0 || i > size(cur)) return;

        node *l, *r, *m, *t;
        split(cur, l, r, i);
```

```
        if (!(!l || size(l) == i)){
            m = l;
            while (m->r) m = m->r;
            long long a = m->a, d = m->d, n = m->n;
            long long c1 = i - (size(l) - m->n), c2 = m->n - c1;

            split(l, l, t, size(l) - m->n);
            pool[idx] = node(a, d, c1);
            merge(l, l, &pool[idx++]);
            pool[idx] = node(a + c1 * d, d, c2);
            merge(l, l, &pool[idx++]);
        }
        merge(cur, l, r);
    }

    inline long long calc_pos(long long n, long long k){
        if (n % k == 0) return 0;
        if ((n - 1) % k == 0) return k - 1;
        long long x = n - (((n / k) * k) + 1);
        return k - 1 - x;
    }

    inline node* erase(node* cur, long long k, long long& pos){
        node *l, *r, *t1, *t2;
        split(cur, l, cur, size(cur->l));
        split(cur, cur, r, cur->n);

        if (l) l = erase(l, k, pos);
        if (pos >= cur->n){
            pos -= cur->n;
            if (r) r = erase(r, k, pos);
            merge(cur, l, r);
            return cur;
        }

        divide(cur, pos);
        split(cur, t1, t2, pos);

        long long n = t2->n;
        pos = calc_pos(n, k);
        pool[idx]= node(t2->a, t2->d * k, ((t2->n - 1) / k) + 1);

        merge(l, l, &pool[idx++]);
        if (r) r = erase(r, k, pos);
        merge(l, l, r);
        return l;
    }

    /// Keep the a, a + k, a + 2*k, ... b terms and delete all the rest
from the segment
    inline void erase(long long a, long long b, long long k){
        node *l, *r, *m;
        long long pos = 0;
        divide(root, a - 1);
```

```
        split(root, l, r, a - 1);
        divide(r, b - a + 1);
        split(r, m, r, b - a + 1);
        m = erase(m, k, pos);
        merge(l, l, m);
        merge(root, l, r);
    }

    inline void erase(long long i){
        node *l, *r, *m;
        divide(root, i - 1);
        split(root, l, r, i - 1);
        divide(r, 1);
        split(r, m, r, 1);
        merge(root, l, r);
    }

    inline node query(long long a, long long b){
        node *l, *r, *m;
        divide(root, a - 1);
        split(root, l, r, a - 1);
        divide(r, b - a + 1);
        split(r, m, r, b - a + 1);
        node res = *m;
        merge(l, l, m);
        merge(root, l, r);
        return res;
    }
};

int main(){
    long long q, n, l, r;

    while (scanf("%lld %lld", &q, &n) != EOF){
        if (q == 0 && n == 0) break;

        Treap T = Treap(n);
        long long total = n;
        while (q--){
            scanf("%lld %lld", &l, &r);
            node res = T.query(l, r);
            printf("%lld %lld\n", res.minval, res.maxval);

            T.erase(l);
            if ((l + 1) <= r) T.erase(l, r - 1, 2);
        }
    }
    return 0;
}
```

Treap With Implicit Keys (Run Length Encoding).cpp

```
---------------------------------------------------
#include <bits/stdtr1c++.h>
```

```
#define MAXN 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

struct node{
   node *l, *r;
   int val, counter, mask, subtree, priority;

   inline node(){
       l = r = 0;
   }

   inline node(int v, int c, int p){
       l = r = 0;
       priority = p;
       subtree = c, val = v, counter = c, mask = (1 << v);
   }

   inline node(int v, int c){
       node(v, c, (rand() << 16) ^ rand());
   }

   inline void update(){
       subtree = counter;
       if (l) subtree += l->subtree;
       if (r) subtree += r->subtree;
   }
} pool[MAXN]; /// Maximum number of nodes in treap

struct Treap{
   int idx;
   struct node* root;

   inline void join(node* cur){
       if (!cur) return;
       cur->update();
       cur->mask = 1 << cur->val;
       if (cur->l) cur->mask |= cur->l->mask;
       if (cur->r) cur->mask |= cur->r->mask;
   }

   inline void merge(node* &cur, node* l, node* r){
       if (!l || !r) cur = l ? l : r;
       else if (l->priority > r->priority) merge(l->r, l->r, r), cur = l;
       else merge(r->l, l, r->l), cur = r;
       if (cur) join(cur);
   }

   /// Smallest v such that l->subtree = v and v >= key
```

```
    inline void split(node* cur, node* &l, node* &r, int key, int counter
= 0){
        if (!cur){
            l = r = 0;
            return;
        }

        int cur_key = counter + (cur->l ? cur->l->subtree : 0);
        if (key <= cur_key) split(cur->l, l, cur->l, key, counter), r =
cur;
        else split(cur->r, cur->r, r, key, cur_key + cur->counter), l =
cur;
        if (cur) join(cur);
    }

    inline void divide(node* &cur, int i){
        node *l, *r, *m, *t;
        split(cur, l, r, i);
        if (!(!l || l->subtree == i)){
            m = l;
            while (m->r) m = m->r;
            int v = m->val, c1 = i - (l->subtree - m->counter), c2 = m-
>counter - c1;

            split(l, l, t, l->subtree - m->counter);
            pool[idx] = node(v, c1);
            merge(l, l, &pool[idx++]); /// Assign to t or m if required
            pool[idx] = node(v, c2);
            merge(l, l, &pool[idx++]); /// Assign to t or m if required

        }
        merge(cur, l, r);
    }

    inline void insert(int i, int v, int c){ /// Inserts c copies of v
after position i
        divide(root, i);
        node *l, *r, *m;
        split(root, l, r, i);
        pool[idx] = node(v, c);
        merge(l, l, &pool[idx++]);
        merge(root, l, r);
    }

    inline void erase(int a, int b){ /// Removes the segment [a:b]
        node *l, *r, *m;
        divide(root, a - 1);
        split(root, l, r, a - 1);
        divide(r, b - a + 1);
        split(r, m, r, b - a + 1);
        merge(root, l, r);
    }
```

```
   inline int query(int a, int b){ /// Number of distinct characters(a-z)
in the segment [a:b]
       node *l, *r, *m;
       divide(root, a - 1);
       split(root, l, r, a - 1);
       divide(r, b - a + 1);
       split(r, m, r, b - a + 1);
       int res = m->mask;
       merge(l, l, m);
       merge(root, l, r);
       return __builtin_popcount(res);
   }

   Treap(){
       idx = 0, root = 0;
   }

   inline int size(){
       if (root) return root->subtree;
       return 0;
   }
};

int main(){

}
```

Treap With Implicit Keys Extended (Run Length Encoding).cpp

```
#include <bits/stdtr1c++.h>

#define MAXN 3000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

struct node{
   bool rev;
   node *l, *r;
   long long pref, suf, res, sum;
   int val, counter, subtree, priority;

   inline node(){
       l = r = 0;
   }

   inline node(int v, int c, int p){
       l = r = 0;
       priority = p;
       subtree = c, counter = c;
       val = pref = suf = res = sum = v, rev = false;
```

```
    }

    inline node(int v, int c){
        node(v, c, (rand() << 16) ^ rand());
    }

    inline void update(){
        subtree = counter;
        if (l) subtree += l->subtree;
        if (r) subtree += r->subtree;
    }
} zero, pool[MAXN]; /// Maximum number of nodes in treap

struct Treap{
    int idx;
    struct node* root;

    inline void push(node* cur){ /// Lazy propagation
        if (cur) cur->update();
        if (cur && cur->rev){
            cur->rev = false;
            if (cur->l) cur->l->rev ^= true;
            if (cur->r) cur->r->rev ^= true;
            swap(cur->l, cur->r);
            swap(cur->pref, cur->suf); /// Exchange maximum prefix sum and
maximum suffix sum because of reversal
        }
    }

    inline void join(node* cur){ /// Update node from its children
        if (!cur) return;

        push(cur);
        node* l = &zero, *r = &zero;
        if (cur->l) push(cur->l), l = cur->l;
        if (cur->r) push(cur->r), r = cur->r;
        long long x = (long long)cur->val * cur->counter;
        long long y = max(x, (long long)cur->val);

        cur->res = y;
        cur->sum = x + l->sum + r->sum;
        cur->suf = (cur->r ? r->suf : y);
        cur->pref = (cur->l ? l->pref : y);
        cur->suf = max(cur->suf, max(r->sum + y, r->sum + x + max(0LL, l-
>suf)));
        cur->pref = max(cur->pref, max(l->sum + y, l->sum + x + max(0LL,
r->pref)));

        if (cur->l) cur->res = max(cur->res, max(l->res, max(0LL, l->suf)
+ y));
        if (cur->r) cur->res = max(cur->res, max(r->res, max(0LL, r->pref)
+ y));
        cur->res = max(cur->res, max(0LL, l->suf) + x + max(0LL, r-
>pref));
```

```
    }

    inline void merge(node* &cur, node* l, node* r){
        push(l), push(r); /// Lazy propagation
        if (!l || !r) cur = l ? l : r;
        else if (l->priority > r->priority) merge(l->r, l->r, r), cur = l;
        else merge(r->l, l, r->l), cur = r;
        if (cur) join(cur);
    }

    /// Smallest v such that l->subtree = v and v >= key
    inline void split(node* cur, node* &l, node* &r, int key, int counter
= 0){
        if (!cur){
            l = r = 0;
            return;
        }

        push(cur); /// Lazy propagation
        int cur_key = counter + (cur->l ? cur->l->subtree : 0);
        if (key <= cur_key) split(cur->l, l, cur->l, key, counter), r =
cur;
        else split(cur->r, cur->r, r, key, cur_key + cur->counter), l =
cur;
        if (cur) join(cur);
    }

    inline void divide(node* &cur, int k){ /// Slices the segment
containing first k numbers
        node *l, *r, *m, *t = 0;
        split(cur, l, r, k);
        if (!(!l || l->subtree == k)){
            m = l;
            while (m->r) m = m->r;
            int v = m->val, c1 = k - (l->subtree - m->counter), c2 = m-
>counter - c1;

            split(l, l, t, l->subtree - m->counter);
            t = &pool[idx++];
            *m = node(v, c1);
            *t = node(v, c2);
            merge(l, l, m);
            merge(l, l, t);

        }
        merge(cur, l, r);
    }

    inline node* build(int i, int j, int* ar){ /// Builds implicit treap
from array in O(n)
        int k = (i + j) >> 1;
        pool[idx] = node(ar[k], 1, (j - i + 1));
        node *cur = &pool[idx++];
        if (i < k) cur->l = build(i, k - 1, ar);
```

```
        if (j > k) cur->r = build(k + 1, j, ar);
        join(cur);
        return cur;
    }

    inline void insert(int i, int v, int c){ /// Inserts c copies of v
after position i
        divide(root, i);
        node *l, *r, *m;
        split(root, l, r, i);
        pool[idx] = node(v, c);
        merge(l, l, &pool[idx++]);
        merge(root, l, r);
    }

    inline void insert(int i, node* cur){ /// Inserts node cur after
position i
        divide(root, i);
        node *l, *r, *m;
        split(root, l, r, i);
        merge(l, l, cur);
        merge(root, l, r);
    }

    inline void erase(int a, int b){ /// Removes the segment [a:b]
        node *l, *r, *m;
        divide(root, a - 1);
        split(root, l, r, a - 1);
        divide(r, b - a + 1);
        split(r, m, r, b - a + 1);
        merge(root, l, r);
    }

    inline void reverse(int a, int b){ /// Reverses the segment [a:b]
        node *l, *r, *m;
        divide(root, a - 1);
        split(root, l, r, a - 1);
        divide(r, b - a + 1);
        split(r, m, r, b - a + 1);
        m->rev ^= true;
        merge(l, l, m);
        merge(root, l, r);
    }

    inline void replace(int a, int b, int c){ /// Replaces all values of
the segment [a:b] with c
        node *l, *r, *m;
        divide(root, a - 1);
        split(root, l, r, a - 1);
        divide(r, b - a + 1);
        split(r, m, r, b - a + 1);
        *m = node(c, b - a + 1);
        merge(l, l, m);
        merge(root, l, r);
```

```
    }

    inline long long getsum(int a, int b){ /// sum of numbers in the
segment [a:b]
        if (a > b) return 0;
        node *l, *r, *m;
        divide(root, a - 1);
        split(root, l, r, a - 1);
        divide(r, b - a + 1);
        split(r, m, r, b - a + 1);
        long long res = m->sum;
        merge(l, l, m);
        merge(root, l, r);
        return res;
    }

    inline long long getmaxsum(int a, int b){ /// Maximum sum of any non-
empty contiguous subsequence of the segment [a:b]
        if (a > b) return 0;
        node *l, *r, *m;
        divide(root, a - 1);
        split(root, l, r, a - 1);
        divide(r, b - a + 1);
        split(r, m, r, b - a + 1);
        long long res = m->res;
        merge(l, l, m);
        merge(root, l, r);
        return res;
    }

    inline long long getmaxsum(){ /// Maximum sum of any non-empty
contiguous subsequence of the array
        return root->res;
    }

    Treap(){
        idx = 0, root = 0;
        zero = node(0, 0);
    }

    inline int size(){
        if (root) return root->subtree;
        return 0;
    }

    inline void dfs(node* cur){
        if (!cur) return;
        push(cur); /// Lazy propagation
        dfs(cur->l);
        for (int i = 0; i < cur->counter; i++) printf("%d ", cur->val);
        dfs(cur->r);
    }
};
```

```
int main(){

}
```

Treap With Implicit Keys.cpp

```
#include <bits/stdtr1c++.h>

#define MAXN 200010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl

using namespace std;

/// Implicit treap for SPOJ Horrible, Range updates and range queries

struct node{
   node *l, *r;
   int subtree, priority;
   long long sum, val, lazy;

   inline node(){
       l = r = 0;
   }

   inline node(long long v, int p){
       l = r = 0;
       priority = p;
       subtree = 1, val = sum = v, lazy = 0;
   }

   inline node(long long v){
       node(v, (rand() << 16) ^ rand());
   }

   inline void update(){
       subtree = 1;
       if (l) subtree += l->subtree;
       if (r) subtree += r->subtree;
   }
} pool[MAXN]; /// Maximum number of nodes in treap

struct Treap{
   int idx;
   struct node* root;

   /// Lazy propagation
   inline void push(node* cur){
       if (!cur || !cur->lazy) return;

       cur->update();
       cur->val += cur->lazy, cur->sum += (cur->lazy * cur->subtree);
       if (cur->l) cur->l->lazy += cur->lazy;
```

```
        if (cur->r) cur->r->lazy += cur->lazy;
        cur->lazy = 0;
    }

    /// Update root node from left child, right child and itself!
    inline void join(node* cur){
        if (!cur) return;

        cur->update();
        cur->sum = cur->val;
        if (cur->l) push(cur->l), cur->sum += cur->l->sum;
        if (cur->r) push(cur->r), cur->sum += cur->r->sum;
    }

    /// Merges two treaps l and r
    inline void merge(node* &cur, node* l, node* r){
        push(l), push(r); /// Lazy propagation
        if (!l || !r) cur = l ? l : r;
        else if (l->priority > r->priority) merge(l->r, l->r, r), cur = l;
        else merge(r->l, l, r->l), cur = r;
        if (cur) join(cur); /// Update root node from left child, right
child and itself!
    }

    /// Splits treap cur, counter is the implicit key on subtree size
    inline void split(node* cur, node* &l, node* &r, int key, int counter
= 0){
        if (!cur){
            l = r = 0;
            return;
        }

        push(cur); /// Lazy propagation
        int cur_key = counter + (cur->l ? cur->l->subtree : 0);
        if (key <= cur_key) split(cur->l, l, cur->l, key, counter), r =
cur;
        else split(cur->r, cur->r, r, key, cur_key + 1), l = cur;
        if (cur) join(cur); /// Update root node from left child, right
child and itself!
    }

    /// Faster insert when appending to the end, appends node v to the end
of the array
    inline void build(int i, int v){
        pool[idx] = node(v);
        merge(root, root, &pool[idx++]);
    }

    /// Builds an implicit treap from an array and returns pointer to the
root in O(n)
    inline node* build(int i, int j, int* ar){
        int k = (i + j) >> 1;
        pool[idx] = node(ar[k], (j - i + 1));
        node *cur = &pool[idx++];
```

```
        if (i < k) cur->l = build(i, k - 1, ar);
        if (j > k) cur->r = build(k + 1, j, ar);
        join(cur);
        return cur;
    }

    /// Inserts a number in the i'th position with value v
    inline void insert(int i, long long v){
        node *l, *r;
        split(root, l, r, i);
        pool[idx] = node(v);
        merge(root, l, &pool[idx++]); /// New node created here
        merge(root, root, r);
    }

    /// Adds v to the segment [a:b]
    inline void update(int a, int b, long long v){
        node *l, *r, *m;
        split(root, l, r, a - 1);
        split(r, m, r, b - a + 1);
        m->lazy += v;
        merge(m, m, r);
        merge(root, l, m);
    }

    /// Returns the sum of the segment[a:b]
    inline long long query(int a, int b){
        node *l, *r, *m;
        split(root, l, r, a - 1);
        split(r, m, r, b - a + 1);

        long long res = m->sum;
        merge(m, m, r);
        merge(root, l, m);
        return res;
    }

    Treap(){
        srand(time(0));
        idx = 0, root = 0;
    }

    inline int size(){
        if (root) return root->subtree;
        return 0;
    }
};

int main(){
    int t, n, q, i, j, k, f, l, r, x, v;

    scanf("%d", &t);
    while (t--){
        Treap T = Treap();
```

```
        scanf("%d %d", &n, &q);
        for (i = 1; i <= n; i++) T.insert(i, 0);

        while (q--){
            scanf("%d %d %d", &f, &l, &r);
            if (!f){
                scanf("%d", &v);
                T.update(l, r, v);
            }
            else printf("%lld\n", T.query(l, r));
        }
    }
    return 0;
}
```

Trie.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define LET 26
#define MAX 1000010
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

int r, idx, counter[MAX], trie[MAX][LET];

void initialize(){
    int i;
    r = 0, idx = 1, counter[r] = 0;
    for (i = 0; i < LET; i++) trie[r][i] = 0;
}

void insert(int x){ /// Set r = 0 before inserting first character
    int i, j;
    if (!trie[r][x]){
        trie[r][x] = idx;
        r = idx++;
        for (i = 0, counter[r] = 1; i < LET; i++) trie[r][i] = 0;
    }
    else r = trie[r][x], counter[r]++;
}

int main(){
    int i, j, x;
    initialize();
    char str[1010];

    while (scanf("%s", str) != EOF){
        r = 0; /// r = root = 0
        for (j = 0; str[j] != 0; j++){
            x = str[j] - 'a';
            insert(x);
```

```
        }
    }
    return 0;
}
```

U128.cpp

```
#include <bits/stdtr1c++.h>

using namespace std;

typedef unsigned long long int U64;

struct U128{
    U64 lo, hi;
    static const U64 bmax = -1;
    static const size_t sz  = 128;
    static const size_t hsz = 64;

    inline U128() : lo(0), hi(0) {}
    inline U128(unsigned long long v) : lo(v), hi(0) {}

    inline U128 operator-() const {
        return ~U128(*this) + 1;
    }

    inline U128 operator~() const {
            U128 t(*this);
            t.lo = ~t.lo;
            t.hi = ~t.hi;
            return t;
      }

    inline U128 &operator +=(const U128 &b) {
        if (lo > bmax - b.lo) ++hi;
        lo += b.lo;
        hi += b.hi;
        return *this;
    }

    inline U128 &operator -= (const U128 &b){
        return *this += -b;
    }

    inline U128 &operator *= (const U128 &b) {
        if (*this == 0 || b == 1) return *this;
        if (b == 0){
            lo = hi = 0;
            return *this;
        }

        U128 a(*this);
        U128 t = b;
        lo = hi = 0;
```

```
        for (size_t i = 0; i < sz; i++) {
            if((t & 1) != 0) *this += (a << i);
            t >>= 1;
        }
        return *this;
    }

    inline U128 &operator /= (const U128 &b) {
            U128 rem;
            divide(*this, b, *this, rem);
            return *this;
    }

    inline U128 &operator %= (const U128 &b) {
            U128 quo;
            divide(*this, b, quo, *this);
            return *this;
    }

    inline static void divide(const U128 &num, const U128 &den, U128 &quo,
U128 &rem) {
        if(den == 0) {
            int a = 0;
            quo = U128(a / a);
        }
        U128 n = num, d = den, x = 1, ans = 0;

        while((n >= d) && (((d >> (sz - 1)) & 1) == 0)) {
            x <<= 1;
            d <<= 1;
        }

        while(x != 0) {
            if(n >= d) {
                n -= d;
                ans |= x;
            }
            x >>= 1, d >>= 1;
        }
        quo = ans, rem = n;
    }

    inline U128 &operator&=(const U128 &b) {
        hi &= b.hi;
        lo &= b.lo;
        return *this;
    }

    inline U128 &operator|=(const U128 &b) {
        hi |= b.hi;
        lo |= b.lo;
        return *this;
    }
```

```
inline U128 &operator<<=(const U128& rhs) {
        size_t n = rhs.to_int();
    if (n >= sz) {
        lo = hi = 0;
        return *this;
    }

    if(n >= hsz) {
        n -= hsz;
        hi = lo;
        lo = 0;
    }

    if(n != 0) {
        hi <<= n;
        const U64 mask(~(U64(-1) >> n));
        hi |= (lo & mask) >> (hsz - n);
        lo <<= n;
    }
    return *this;
   }

inline U128 &operator>>=(const U128& rhs) {
        size_t n = rhs.to_int();
    if (n >= sz) {
        lo = hi = 0;
        return *this;
    }

    if(n >= hsz) {
        n -= hsz;
        lo = hi;
        hi = 0;
    }

    if(n != 0) {
        lo >>= n;
        const U64 mask(~(U64(-1) << n));
        lo |= (hi & mask) << (hsz - n);
        hi >>= n;
    }

    return *this;
   }

inline int to_int() const { return static_cast<int> (lo); }
inline U64 to_U64() const { return lo; }

inline bool operator == (const U128 &b) const {
    return hi == b.hi && lo == b.lo;
}

inline bool operator != (const U128 &b) const {
```

```
        return !(*this == b);
    }

    inline bool operator < (const U128 &b) const {
        return (hi == b.hi) ? lo < b.lo : hi < b.hi;
    }

    inline bool operator >= (const U128 &b) const {
        return ! (*this < b);
    }

    inline U128 operator & (const U128 &b) const {
        U128 a(*this); return a &= b;
    }

    inline U128 operator << (const U128 &b) const {
        U128 a(*this); return a <<= b;
    }

    inline U128 operator >> (const U128 &b) const {
        U128 a(*this); return a >>= b;
    }

    inline U128 operator * (const U128 &b) const {
        U128 a(*this); return a *= b;
    }

    inline U128 operator + (const U128 &b) const {
        U128 a(*this); return a += b;
    }

    inline U128 operator - (const U128 &b) const {
        U128 a(*this); return a -= b;
    }

    inline U128 operator % (const U128 &b) const {
        U128 a(*this); return a %= b;
    }

    inline void print(){
        U128 x = *this;
        char str[128];
        int i, j, len = 0;

        do{
            str[len++] = (x % 10).lo + 48;
            x /= 10;
        } while (x != 0);

        reverse(str, str + len);
        str[len] = 0;
        puts(str);
    }
};
```

```
inline U128 gcd(U128 a, U128 b){
   if (b == 0) return a;
   return gcd(b, a % b);
}

inline U128 expo(U128 b, U128 e){
   U128 res = 1;
   while (e != 0){
       if ((e & 1) != 0) res *= b;
       e >>= 1, b *= b;
   }
   return res;
}

inline U128 expo(U128 x, U128 n, U128 m){
   U128 res = U128(1);
   while (n != 0){
       if ((n & 1) != 0){
           res *= n;
           res %= m;
       }
       x *= x;
       x %= m;
       n >>= 1;
   }
   return res % m;
}

struct Rational{
   U128 p, q;

   inline Rational(){
       p = 0, q = 1;
   }

   inline Rational(U128 P, U128 Q) : p(P), q(Q){
       simplify();
   }

   inline void simplify() {
       U128 g = gcd(p, q);
       p /= g;
       q /= g;
   }

   inline Rational operator+ (const Rational &f) const {
       return Rational(p * f.q + q * f.p, q * f.q);
   }
   inline Rational operator- (const Rational &f) const {
       return Rational(p * f.q - q * f.p, q * f.q);
   }
   inline Rational operator* (const Rational &f) const {
       return Rational(p * f.p, q * f.q);
```

```
    }
    inline Rational operator/ (const Rational &f) const {
        return Rational(p * f.q, q * f.p);
    }
};

int main(){
    U128 X = U128(9178291938173ULL);
    U128 Y = U128(123456789123456ULL);
    U128 M = U128(1000000000000000000ULL);

    for (int i = 0; i < 10000; i++){
        U128 R = expo(X, Y, M);
    }
    return 0;
}
```

Walsh-Hadamard Transformation.c

```
--------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 1048576
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

long long ar[MAX];

void walsh_transform(long long* ar, int n){
    if (n == 0) return;

    int i, m = n / 2;
    walsh_transform(ar, m);
    walsh_transform(ar + m, m);

    for (i = 0; i < m; i++){
        long long x = ar[i], y = ar[i + m];
        ar[i] = x + y, ar[i + m] = x - y;
    }
}

void inverse_walsh_transform(long long* ar, int n){
    if (n == 0) return;

    int i, m = n / 2;
    inverse_walsh_transform(ar, m);
    inverse_walsh_transform(ar + m, m);

    for (i = 0; i < m; i++){
        long long x = ar[i], y = ar[i + m];
        ar[i] = (x + y) >> 1, ar[i + m] = (x - y) >> 1;
    }
}
```

```
int main(){
   int n, i, j, k, x;

   scanf("%d", &n);
   while (n--){
       scanf("%d", &x);
       ar[x]++;
   }

   walsh_transform(ar, MAX);
   for (i = 0; i < MAX; i++) ar[i] *= ar[i];
   inverse_walsh_transform(ar, MAX);

   long long res = 0;
   for (i = 0; i < MAX; i++) res += (ar[i] * i);
   printf("%lld\n", res / 2);
   return 0;
}
```

Walsh-Hadamard Transformation.cpp

```
--------------------------------------------------
#include <bits/stdtr1c++.h>

#define MAX 1048576
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x << endl
#define ran(a, b) ((((rand() << 15) ^ rand()) % ((b) - (a) + 1)) + (a))

using namespace std;

/// Fast Walsh-Hadamard Transformation in n log n
namespace fwht{ /// hash = 840614
   const int OR = 0;
   const int AND = 1;
   const int XOR = 2;

   long long P1[MAX], P2[MAX];

   void walsh_transform(long long* ar, int n, int flag = XOR){
       if (n == 0) return;

       int i, m = n / 2;
       walsh_transform(ar, m, flag);
       walsh_transform(ar + m, m, flag);

       for (i = 0; i < m; i++){ /// Don't forget modulo if required
           long long x = ar[i], y = ar[i + m];
           if (flag == OR) ar[i] = x, ar[i + m] = x + y;
           if (flag == AND) ar[i] = x + y, ar[i + m] = y;
           if (flag == XOR) ar[i] = x + y, ar[i + m] = x - y;
       }
   }
```

```
    void inverse_walsh_transform(long long* ar, int n, int flag = XOR){
        if (n == 0) return;

        int i, m = n / 2;
        inverse_walsh_transform(ar, m, flag);
        inverse_walsh_transform(ar + m, m, flag);

        for (i = 0; i < m; i++){ /// Don't forget modulo if required
            long long x = ar[i], y = ar[i + m];
            if (flag == OR) ar[i] = x, ar[i + m] = y - x;
            if (flag == AND) ar[i] = x - y, ar[i + m] = y;
            if (flag == XOR) ar[i] = (x + y) >> 1, ar[i + m] = (x - y) >>
1; /// Modular inverse if required here
        }
    }

    vector <long long> convolution(int n, long long* A, long long* B, int
flag = XOR){
        assert(__builtin_popcount(n) == 1); /// n must be a power of 2
        for (int i = 0; i < n; i++) P1[i] = A[i];
        for (int i = 0; i < n; i++) P2[i] = B[i];

        walsh_transform(P1, n, flag);
        walsh_transform(P2, n, flag);
        for (int i = 0; i < n; i++) P1[i] = P1[i] * P2[i];
        inverse_walsh_transform(P1, n, flag);
        return vector<long long> (P1, P1 + n);
    }

    /// For i = 0 to n - 1, j = 0 to n - 1
    /// v[i or j] += A[i] * B[j]
    vector <long long> or_convolution(int n, long long* A, long long* B){
        return convolution(n, A, B, OR);
    }

    /// For i = 0 to n - 1, j = 0 to n - 1
    /// v[i and j] += A[i] * B[j]
    vector <long long> and_convolution(int n, long long* A, long long* B){
        return convolution(n, A, B, AND);
    }

    /// For i = 0 to n - 1, j = 0 to n - 1
    /// v[i xor j] += A[i] * B[j]
    vector <long long> xor_convolution(int n, long long* A, long long* B){
        return convolution(n, A, B, XOR);
    }
}

int n;
long long A[MAX], B[MAX], C[MAX];

void brute(){
    int i, j;
```

```
    clr(C);
    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            C[i ^ j] += (A[i] * B[j]);
        }
    }
}

int main(){
    int i, j, k, l, x;

    n = 1 << 15;
    for (i = 0; i < n; i++) A[i] = ran(1, 1000000);
    for (i = 0; i < n; i++) B[i] = ran(1, 1000000);
    brute();
    unsigned long long h1 = 0, h2 = 0;
    for (i = 0; i < n; i++) h1 = (h1 * 1000000007) + C[i] + 97;

    vector <long long> v = fwht::xor_convolution(n, A, B);
    for (i = 0; i < n; i++) h2 = (h2 * 1000000007) + v[i] + 97;

    dbg(h1);
    dbg(h2);
    return 0;
}
```

Z Algorithm.c

```
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

#define MAX 100010
#define min(a,b) ((a)<(b) ? (a):(b))
#define max(a,b) ((a)>(b) ? (a):(b))
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)

char str[MAX];
int n, Z[MAX];

void ZFunction(){ /// Z[i] = lcp of the suffix starting from i with str
    int i, j, k, l, r, p;
    Z[0] = n, l = 0, r = 0;
    for (i = 1; i < n; i++){
        if (i > r){
            k = 0;
            while ((i + k) < n && str[i + k] == str[k]) k++;
            Z[i] = k;
            if (Z[i]) l = i, r = i + Z[i] - 1;
        }
        else{
            p = i - l;
            if (Z[p] < (r - i + 1)) Z[i] = Z[p];
```

```
            else{
                k = r + 1;
                while (k < n && str[k - i] == str[k]) k++;
                l = i, r = k - 1;
                Z[i] = (r - l + 1);
            }
        }
    }
}

/// Z[i] = lcp of the suffix starting from i with str
void ZFunction(char* str){ /// hash = 998923
   int i, l, r, x;

   l = 0, r = 0;
   for (i = 1; str[i]; i++){
       Z[i] = max(0, min(Z[i - l], r - i));
       while (str[i + Z[i]] && str[Z[i]] == str[i + Z[i]]) Z[i]++;
       if ((i + Z[i]) > r) l = i, r = i + Z[i];
   }
   Z[0] = i;
}

int main(){
   scanf("%s", str);
   n = strlen(str);
   ZFunction();
   return 0;
}
```