CODE LIBRARY

Contents

1 Data Structure

1.1 2D BIT

```

// 2D BIT
int max_x, max_y, tree[MAX+10][MAX+10]; //An array, suppose arr[MAX][MAX]
// 1 based indexing

void update(int x , int y , int val) //Updating arr[x][y]
{
    int y1;
    while (x <= max_x)
    {
        y1 = y;
        while (y1 <= max_y)
        {
            tree[x][y1] += val;
            y1 += (y1 & -y1);
        }
        x += (x & -x);
    }
}

int query(int x , int y) // Cumulative sum from arr[1][1] to arr[x][y]
{
    int y1, ret = 0;
    while (x)
    {
        y1 = y;
        while (y1)
        {
            ret += tree[x][y1];
            y1 -= (y1 & -y1);
        }
        x -= (x & -x);
    }
    return ret;
}
```

1.2 2D Segment Tree RMQ

```

//2D Segment Tree

const int inf = 1000000000;
struct segTree{
    int arr[MAX<<2];

    segTree(){
        for(int i = 0; i < (MAX << 2); i++) arr[i] = inf;
    }
```

```
11
12      void update(int idx, int st, int ed, int pos, int val, vector<int> &
            nodeList){
13          nodeList.push_back(idx);
14
15          if(st == ed){
16              arr[idx] = val;
17              return;
18          }
19
20          int mid = (st+ed)/2, l = idx << 1, r = l | 1;
21          if(pos <= mid) update(l, st, mid, pos, val, nodeList);
22          else update(r, mid+1, ed, pos, val, nodeList);
23
24          arr[idx] = min(arr[l], arr[r]);
25      }
26
27      int query(int idx, int st, int ed, int i, int j)
28      {
29          if(st == i && ed == j) return arr[idx];
30
31          int mid = (st+ed)/2, l = idx << 1, r = l | 1;
32          if(j <= mid) return query(l, st, mid, i, j);
33          if(i > mid) return query(r, mid+1, ed, i, j);
34          else return min(query(l, st, mid, i, mid), query(r, mid+1, ed, mid+1,
                j));
35      }
36  };
37
38  struct _2DsegTree{
39      segTree segArr[MAX<<2];
40      vector<int> affected_nodes;
41
42      void update(int idx, int st, int ed, int i, int j, int val){
43          if(st == ed){
44              affected_nodes.clear();
45              segArr[idx].update(1, 1, MAX, j, val, affected_nodes);
46              return;
47          }
48
49          int mid = (st+ed)/2, l = idx << 1, r = l|1;
50          if(i <= mid) update(l, st, mid, i, j, val);
51          else update(r, mid+1, ed, i, j, val);
52
53          for(int p = 0; p < affected_nodes.size(); p++)
54          {
55              int q = affected_nodes[p];
56              segArr[idx].arr[q] = min(segArr[l].arr[q], segArr[r].arr[q]);
57          }
58      }
59
```

```
    int query(int idx, int st, int ed, int st_r, int ed_r, int st_c, int ed_c)
        {
        assert(st_r <= ed_r && st_c <= ed_c);

        if(st == st_r && ed == ed_r) return segArr[idx].query(1, 1, MAX, st_c,
            ed_c);

        int mid = (st+ed)/2, l = idx << 1, r = l | 1;
        if(ed_r <= mid) return query(l, st, mid, st_r, ed_r, st_c, ed_c);
        else if(st_r > mid) return query(r, mid+1, ed, st_r, ed_r, st_c, ed_c)
            ;
        return min(query(l, st, mid, st_r, mid, st_c, ed_c), query(r, mid+1,
            ed, mid+1, ed_r, st_c, ed_c));
    }
};
```

1.3 AHO CHORASIC DISTRIBUTED FREQUENCY

```
#include<bits/stdc++.h>
using namespace std;
#define D(x)    cout << #x " = " << (x) << endl
#define MAX     1000000

#define ALPHABET_SIZE   26

struct aho_corasick{

    vector< int > visit;
    vector < vector < int > > f_tree;
    vector < int > patMark;
    vector < int > ocr;
    vector < int > dp;

    struct trie{
        int fail, idx;
        vector<int> endCounter;
        trie *nxt[ALPHABET_SIZE];
        int fail_nxt[ALPHABET_SIZE];

        trie(int _idx){
            fail = 0;
            idx = _idx;
            memset(nxt, 0, sizeof(nxt));
            memset(fail_nxt, -1,sizeof(fail_nxt));
        }
    };

    vector< trie * > node;

    aho_corasick(){
        node.push_back(new trie(0));
        visit.push_back(0);
```

```
35          f_tree.push_back(vector<int>());
36          dp.push_back(-1);
37      }
38
39      int rank(char ch){return ch - 'a';}
40
41      void insert(char *pat, int number){ ///NUMBERS MUST START FROM 0 AND
            INCREASE BY 1
42          trie *cur = node[0];
43          for(int i = 0; pat[i]; i++){
44              int v = rank(pat[i]);
45              if(!cur->nxt[v]){
46                  cur->nxt[v] = new trie(node.size());
47                  node.push_back(cur->nxt[v]);
48                  visit.push_back(0);
49                  f_tree.push_back(vector<int>());
50                  dp.push_back(-1);
51              }
52              cur = cur->nxt[v];
53          }
54          cur->endCounter.push_back(number);
55          patMark.push_back(cur->idx);
56          ocr.push_back(0);
57      }
58
59      void build(){
60          int i;
61          queue<trie *> Q;
62
63          for(i = 0; i < ALPHABET_SIZE; i++)
64              if(node[0] -> nxt[i]){
65                  trie *p = node[0] -> nxt[i];
66                  p -> fail = 0;
67                  f_tree[0].push_back(p->idx);
68                  Q.push(p);
69              }
70
71          while(!Q.empty()){
72              trie *u = Q.front(); Q.pop();
73              for(i = 0; i < (int) ALPHABET_SIZE; i++){
74                  if(u -> nxt[i]){
75                      int f = u -> idx;
76                      trie *v = u -> nxt[i];
77
78                      while(true){
79                          f = node[f] -> fail;
80                          if(node[f] -> nxt[i]) {
81                              v -> fail = node[f] -> nxt[i] -> idx;
82                              break;
83                          }
84                          else if(!f) break;
```

```
                    }

                    f_tree[v -> fail].push_back( v -> idx);
                    Q.push(u->nxt[i]);
                }
            }
        }
    }

    vector<int> nodeList;
    int dfs(int idx){
        if(dp[idx] != -1) return dp[idx];
        nodeList.push_back(idx);

        int ret = visit[idx];

        for(int i = 0; i < (int) f_tree[idx].size(); i++)
            ret += dfs(f_tree[idx][i]);

        return dp[idx] = ret;

    }

    inline int transition(int idx, int v){
        if(node[idx] -> fail_nxt[v] != -1) return node[idx] -> fail_nxt[v];
        if(node[idx] -> nxt[v]) return node[idx] -> fail_nxt[v] = node[idx] ->
            nxt[v] -> idx;
        int f = node[idx] -> fail;
        if(node[f] -> nxt[v]) return node[idx] -> fail_nxt[v] = node[f] -> nxt
            [v] -> idx;
        if(!f) return node[idx] -> fail_nxt[v] = node[f] -> idx;
        return node[idx] -> fail_nxt[v] = transition(f, v);
    }

    vector < int > path;
    void traverse(char *text){
        assert(path.empty());

        trie *current = node[0];
        path.push_back(0);

        for(int i = 0; text[i]; i++){
            int v = rank(text[i]);
            current = node[transition(current->idx, v)];
            path.push_back(current -> idx);
        }

        for(int i = 0; i < (int) path.size(); i++) visit[path[i]]++;

        for(int i = 0; i < (int) patMark.size(); i++)
            ocr[i] = dfs(patMark[i]);
```

```
134
135         for(int i = 0; i < (int) nodeList.size(); i++) dp[nodeList[i]] = -1;
136         nodeList.clear();
137
138         for(int i = 0; i < (int) path.size(); i++) visit[path[i]]--;
139         path.clear();
140     }
141
142     void clear(){
143         for(int i = 0; i < (int) node.size(); i++) { node[i] -> endCounter.
                clear(); delete(node[i]); }
144         node.clear();
145         visit.clear();
146         f_tree.clear();
147         patMark.clear();
148         ocr.clear();
149         dp.clear();
150
151         node.push_back(new trie(0));
152         visit.push_back(0);
153         f_tree.push_back(vector<int>());
154         dp.push_back(-1);
155     }
156
157 };
158
159 aho_corasick ac;
160 char txt[MAX+5], pat[MAX+5];
161
162 int main()
163 {
164     //freopen("in.txt", "r", stdin);
165
166     int i, n, t, cs;
167
168     scanf("%d", &t);
169     for(cs = 1; cs <= t; cs++){
170         scanf("%d", &n);
171         scanf("%s", txt);
172         for(i = 0; i < n; i++){
173             scanf("%s", pat);
174             ac.insert(pat, i);
175         }
176         ac.build();
177         ac.traverse(txt);
178         printf("Case %d:\n", cs);
179         for(i = 0; i < n; i++) printf("%d\n", ac.ocr[i]);
180
181         ac.clear();
182     }
183     return 0;
```

```
}
```

1.4 AHO CHORASIC DYNAMIC ONLINE

```

#include<bits/stdc++.h>
using namespace std;
#define D(x)     cout << #x " = " << (x) << endl
#define MAX      300000
typedef long long int LL;

#define ALPHABET_SIZE   26
#define MAXL            300000

struct aho_corasick{

    struct trie{
        int fail, idx, endCounter, dp;
        trie *nxt[ALPHABET_SIZE];

        trie(int _idx){
            fail = 0;
            idx = _idx;
            endCounter = dp = 0;
            memset(nxt, 0, sizeof(nxt));
        }
    };

    vector< trie * > node;

    aho_corasick(){
        node.push_back(new trie(0));
    }

    int rank(char ch){return ch - 'a';}

    void insert(char *pat, int sign){
        trie *cur = node[0];
        for(int i = 0; pat[i]; i++){
            int v = rank(pat[i]);
            if(!cur->nxt[v]){
                cur->nxt[v] = new trie(node.size());
                node.push_back(cur->nxt[v]);
            }
            cur = cur->nxt[v];
        }
        assert(sign == -1 || sign == +1);
        cur->endCounter += sign;
    }

    void build(){
        int i;
```

```
49          queue<trie *> Q;
50
51          for(i = 0; i < ALPHABET_SIZE; i++)
52              if(node[0] -> nxt[i]){
53                  trie *p = node[0] -> nxt[i];
54                  p -> fail = 0;
55                  p -> dp = p -> endCounter;
56                  Q.push(p);
57              }
58
59          while(!Q.empty()){
60              trie *u = Q.front(); Q.pop();
61              for(i = 0; i < (int) ALPHABET_SIZE; i++){
62                  if(u -> nxt[i]){
63                      int f = u -> idx;
64                      trie *v = u -> nxt[i];
65
66                      while(true){
67                          f = node[f] -> fail;
68                          if(node[f] -> nxt[i]) {
69                              v -> fail = node[f] -> nxt[i] -> idx;
70                              break;
71                          }
72                          else if(!f) break;
73                      }
74
75                      Q.push(u->nxt[i]);
76                      v -> dp = v -> endCounter + node[v -> fail] -> dp;
77                  }
78              }
79          }
80      }
81
82
83      LL traverse(char *text){
84          LL ret = 0;
85          trie *current = node[0];
86
87          for(int i = 0; text[i]; i++){
88              int v = rank(text[i]);
89              if(current -> nxt[v]) current = current -> nxt[v];
90              else{
91                  int f = current -> idx;
92                  while(true){
93                      f = node[f] -> fail;
94                      if(node[f] -> nxt[v]){
95                          current = node[f] -> nxt[v];
96                          break;
97                      }
98                      else if(!f) {current = node[f]; break;}
99                  }
```

```
            }

            ret += (current -> dp);//path.push_back(current -> idx);
        }
        return ret;
    }

    void clear(){
        for(int i = 0; i < (int) node.size(); i++) { delete(node[i]); }
        node.clear();
        node.push_back(new trie(0));
    }
};

#define LOG     23
struct dynamic_ac{
    aho_corasick ac[LOG];
    vector< pair<string,int> > input[LOG];

    void insert(char *str, int sign){
        int i, k;
        for(k = 0; k < LOG; k++) if(input[k].size() == 0) break;
        input[k].push_back(make_pair(string(str), sign));
        ac[k].insert(str, sign);
        for(i = 0; i < k; i++){
            ac[i].clear();
            for(auto s : input[i]) {
                ac[k].insert((char *) s.first.c_str(), s.second);
                input[k].push_back(s);
            }
            input[i].clear();
        }
        ac[k].build();
    }

    LL query(char *str){
        LL ret = 0;
        for(int i = 0; i < LOG; i++)
                ret += ac[i].traverse(str);

        return ret;
    }
} d_ac;

char str[MAX+5];

int main()
{
//    freopen("in.txt", "r", stdin);

    int q, tp;
```

```
    scanf("%d", &q);

    while(q--){
        scanf("%d", &tp);
        scanf("%s", str);

        if(tp == 1) d_ac.insert(str, 1);
        else if(tp == 2) d_ac.insert(str, -1);
        else {printf("%lld\n", d_ac.query(str)); fflush(stdout);}
    }
    return 0;
}
```

1.5 BASIC DSU

```
int color[MAX+5];
vector<int> edge[MAX+5];
int subsize[MAX+5];

int init(int idx, int p = -1){
    int ret = 1;
    for(auto x : edge[idx])
        if(x != p)
            ret += init(x, idx);
    return subsize[idx] = ret;
}

int freq[MAX+5];
LL sum[MAX+5];
/*
    freq[i] = number of times the color 'i' appeared
    sum[i] = sum of colors which appeared 'i' times
*/

void insert_all(int idx, int p){
    sum[freq[color[idx]]] -= color[idx];
    freq[color[idx]]++;
    sum[freq[color[idx]]] += color[idx];

    for(auto x : edge[idx])
        if(x != p)
            insert_all(x, idx, r);
}

void remove_all(int idx, int p){
    sum[freq[color[idx]]] -= color[idx];
    freq[color[idx]]--;
    sum[freq[color[idx]]] += color[idx];

    for(auto x : edge[idx])
        if(x != p)
            remove_all(x, idx);
```

```
}

void dfs(int idx, int p, bool keep){ ///returns the maximum frequency
    int bigchild = -1, mx = 0, ret = 0;
    for(auto x : edge[idx])
        if(x != p && subsize[x] > mx){
            mx = subsize[x];
            bigchild = x;
        }

    for(auto x : edge[idx])
        if(x != p && x != bigchild)
            dfs(x, idx, false);

    if(bigchild != -1) dfs(bigchild, idx, true);
    for(auto x : edge[idx])
        if(x != p && x != bigchild)
            insert_all(x, idx, ret);

    ///INSERTING IDX
    sum[freq[color[idx]]] -= color[idx];
    freq[color[idx]]++;
    sum[freq[color[idx]]] += color[idx];

    ///Answer your queries here
    if(!keep) remove_all(idx, p);
}
```

1.6 BIT

```

void update(int idx, int val)
{
    while(idx <= mxval)
        BIT[idx] += val, idx += idx&-idx;
}

LL query(int idx)
{
    LL ret = 0;
    while(idx)
        ret += BIT[idx], idx -= idx&-idx;
    return ret;
}
```

1.7 HashMap

```

struct hashMap{
    int t, n;
    int id[MAX], value[MAX];
    char name[MAX][11];
```

```
6
7     hashMap(){
8         t = 1;
9         n = 9999991;
10    }
11
12    void clear(){
13        t++;
14    }
15
16    int getHash(char *str){
17        LL ret = 0;
18        int i;
19
20        for(i = 0; str[i] ; i++){
21            ret = (ret * hp) % n;
22            ret = ret + (str[i]);
23        }
24
25        return ret;
26    }
27
28    void add(char *str, int marks){
29        int x = getHash(str);
30
31        while(id[x] == t){
32            if(strcmp(name[x], str) == 0){
33                value[x] += marks;
34                return;
35            }
36            x = (x + 1) % n;
37        }
38
39        id[x] = t;
40        strcpy(name[x], str);
41        value[x] = marks;
42
43        return;
44    }
45
46    int query_index(char *str){
47        int x = getHash(str);
48
49        while(id[x] == t){
50            if(strcmp(name[x], str) == 0)
51                return x;
52            x = (x + 1) % n;
53        }
54        return -1;
55    }
56
```

```
    int query(char *str){
        int p = query_index(str);
        if(p == -1) return 0;
        return value[p];
    }

    void erase(char *str){
        int p = query_index(str);
        if(p == -1) return;
        return (void) (value[p] = 0);
    }
} HM;
```

1.8 LCT ROOTED TREE

```

#include<bits/stdc++.h>
using namespace std;
#define D(x)    cout << #x " = " << (x) << endl
#define MAX     300000

int lct_par[MAX+5];
int n;

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

void normalize(Node *x){ ///PUSH THE LAZY DOWN
    if(x->lazy){
        if(x->l){
            x->l->lazy += x->lazy;
            x->l->value += x->lazy;
        }
        if(x->r){
            x->r->lazy += x->lazy;
            x->r->value += x->lazy;
        }

        x->lazy = 0;
    }
}

void update(Node *x)
{
```

```
38      assert(!x->lazy);
39      x->sz = 1;
40      if(x->l) x->sz += x->l->sz;
41      if(x->r) x->sz += x->r->sz;
42  }
43
44  void set_value(Node *x, int v){
45      x->value = v;
46  }
47
48  void rotr(Node *x)
49  {
50      Node *y, *z;
51      y = x->p, z = y->p;
52      normalize(y);
53      normalize(x);
54
55      if((y->l = x->r)) y->l->p = y;
56      x->r = y, y->p = x;
57      if((x->p = z))
58      {
59          if(y == z->l) z->l = x;
60          else z->r = x;
61      }
62      x->pp = y->pp;
63      y->pp = 0;
64      update(y);
65  }
66
67  void rotl(Node *x)
68  {
69      Node *y, *z;
70      y = x->p, z = y->p;
71      normalize(y);
72      normalize(x);
73
74      if((y->r = x->l)) y->r->p = y;
75      x->l = y, y->p = x;
76      if((x->p = z))
77      {
78          if(y == z->l) z->l = x;
79          else z->r = x;
80      }
81      x->pp = y->pp;
82      y->pp = 0;
83      update(y);
84  }
85
86  void splay(Node *x)
87  {
88      Node *y, *z;
```

```
    while(x->p)
    {
        y = x->p;
        if(y->p == 0)
        {
            if(x == y->l) rotr(x);
            else rotl(x);
        }
        else
        {
            z = y->p;
            if(y == z->l)
            {
                if(x == y->l) rotr(y), rotr(x);
                else rotl(x), rotr(x);
            }
            else
            {
                if(x == y->r) rotl(y), rotl(x);
                else rotr(x), rotl(x);
            }
        }
    }
    normalize(x);
    update(x);
}

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

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

```
140         x->pp = 0;
141         update(y);
142         splay(x);
143     }
144     return last;
145 }
146 
147 Node *root(Node *x)
148 {
149     access(x);
150     while(x->l)
151     {
152         x = x->l;
153     }
154     splay(x);
155     return x;
156 }
157 
158 void cut(Node *x)
159 {
160     access(x);
161     x->l->p = 0;
162     x->l = 0;
163     update(x);
164 }
165 
166 void link(Node *x, Node *y)
167 {
168     access(x);
169     access(y);
170     x->l = y;
171     y->p = x;
172     update(x);
173 }
174 
175 Node *lca(Node *x, Node *y)
176 {
177     access(x);
178     return access(y);
179 }
180 
181 int depth(Node *x)
182 {
183     access(x);
184     return x->sz - 1;
185 }
186 
187 int query(Node *x){
188     access(x);
189     return x->value;
190 }
```

```
191
192 void range_update(Node *x, int lz){
193     access(x);
194     x->lazy += lz;
195     x->value += lz;
196 }
197
198 int special_root(Node *x){
199     access(x);
200     Node *r = root(x);
201     splay(r);
202     r = r->r;
203     while(r->l) r = r->l;
204     return r->label;
205 }
206
207 class LinkCut
208 {
209 public:
210     Node *x;
211
212     LinkCut(int n)
213     {
214         x = new Node[n+5];
215         for(int i = 1; i <= n; i++)
216         {
217             x[i].label = i;
218             update(&x[i]);
219         }
220     }
221
222     virtual ~LinkCut()
223     {
224         delete[] x;
225     }
226
227     void set_value(int u, int v){
228         ::set_value(&x[u], v);
229     }
230
231     void link(int u, int v)
232     {
233         lct_par[u] = v;
234
235         int sz = ::query(&x[u]);
236         ::range_update(&x[v], +sz);
237
238         assert(u);
239         assert(v);
240         ::link(&x[u], &x[v]);
241     }
```

```
242
243     void cut(int u)
244     {
245         assert(u);
246         int sz = ::query(&x[u]);
247         if(lct_par[u]){
248             ::range_update(&x[lct_par[u]], -sz);
249         }
250         ::cut(&x[u]);
251     }
252
253     int root(int u)
254     {
255         int ret = ::root(&x[u])->label;
256         if(ret <= n) return ret;
257         return ::special_root(&x[u]);
258     }
259
260     int depth(int u)
261     {
262         return ::depth(&x[u]);
263     }
264
265     int lca(int u, int v)
266     {
267         return ::lca(&x[u], &x[v])->label;
268     }
269
270     int query(int u){
271         return ::query(&x[u]);
272     }
273 }*lctree;
274
275 #define BLACK       0
276 #define WHITE       1
277
278 vector<int> edge[MAX+5];
279 int same[MAX+5], dif[MAX+5], running, par[MAX+5], color[MAX+5];
280
281 int dfs(int idx, int p = -1){
282     assert(idx);
283
284     same[idx] = ++running;
285     dif[idx] = ++running;
286     color[idx] = BLACK;
287     par[idx] = p;
288
289
290     int ret = 1;
291     for(auto x : edge[idx])
292         if(x != p)
```

```
            ret += dfs(x, idx);

    if(p != -1) lctree -> link(idx, same[p]);
    lctree->link(same[idx], idx);


    lctree->set_value(idx, ret);
    lctree->set_value(same[idx], ret - 1);
    return ret;
}

void toggle(int idx){

    lctree->cut(same[idx]);
    lctree->link(dif[idx], idx);

    int p = par[idx];

    if(p != -1){
        if(color[par[idx]] == color[idx]){
            lctree-> cut(idx);
            lctree ->link(idx, dif[par[idx]]);
        }
        else{
            lctree-> cut(idx);
            lctree->link(idx, same[par[idx]]);
        }
    }

    swap(same[idx], dif[idx]);
    color[idx] = 1 - color[idx];

}

bool vis[MAX+5];
int brute(int idx){
    if(vis[idx]) return 0;
    int ret = 1;

    vis[idx] = true;
    for(auto x: edge[idx])
        if(color[x] == color[idx])
            ret += brute(x);

    return ret;
}

int main(){
    //freopen("in.txt", "r", stdin);

    int i, u, v, q, tp, idx, ans;
```

```
    scanf("%d", &n);
    lctree = new LinkCut(n * 3);
    running = n;

    for(i = 1; i < n; i++){
        scanf("%d %d", &u, &v);
        edge[u].push_back(v);
        edge[v].push_back(u);
    }

    dfs(1);


    scanf("%d", &q);
    while(q--){
        scanf("%d %d", &tp, &idx);
        if(tp == 0){
            printf("%d\n", ans = lctree->query(lctree->root(idx)));
            //memset(vis, false, sizeof(vis));
            //assert(ans == brute(idx));
        }
        else{
            toggle(idx);
        }
    }
    return 0;
}
```

1.9 LCT UNROOTED TREE

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


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

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

```
23
24  void normalize(Node *x){ ///PUSH THE LAZY DOWN
25      if(x->flip){
26          if(x->l){
27              x->l->flip ^= 1;
28              swap(x->l->l, x->l->r);
29          }
30          if(x->r){
31              x->r->flip ^= 1;
32              swap(x->r->l, x->r->r);
33          }
34          x->flip = 0;
35      }
36  }
37
38  void rotr(Node *x)
39  {
40      Node *y, *z;
41      y = x->p, z = y->p;
42      normalize(y);
43      normalize(x);
44      if((y->l = x->r)) y->l->p = y;
45      x->r = y, y->p = x;
46      if((x->p = z))
47      {
48          if(y == z->l) z->l = x;
49          else z->r = x;
50      }
51      x->pp = y->pp;
52      y->pp = 0;
53      update(y);
54  }
55
56  void rotl(Node *x)
57  {
58      Node *y, *z;
59      y = x->p, z = y->p;
60      normalize(y);
61      normalize(x);
62
63      if((y->r = x->l)) y->r->p = y;
64      x->l = y, y->p = x;
65      if((x->p = z))
66      {
67          if(y == z->l) z->l = x;
68          else z->r = x;
69      }
70      x->pp = y->pp;
71      y->pp = 0;
72      update(y);
73  }
```

```
74
75  void splay(Node *x)
76  {
77      Node *y, *z;
78      while(x->p)
79      {
80          y = x->p;
81          if(y->p == 0)
82          {
83              if(x == y->l) rotr(x);
84              else rotl(x);
85          }
86          else
87          {
88              z = y->p;
89              if(y == z->l)
90              {
91                  if(x == y->l) rotr(y), rotr(x);
92                  else rotl(x), rotr(x);
93              }
94              else
95              {
96                  if(x == y->r) rotl(y), rotl(x);
97                  else rotr(x), rotl(x);
98              }
99          }
100     }
101     normalize(x);
102     update(x);
103 }
104
105 Node *access(Node *x)
106 {
107     splay(x);
108     if(x->r)
109     {
110         x->r->pp = x;
111         x->r->p = 0;
112         x->r = 0;
113         update(x);
114     }
115
116     Node *last = x;
117     while(x->pp)
118     {
119         Node *y = x->pp;
120         last = y;
121         splay(y);
122         if(y->r)
123         {
124             y->r->pp = y;
```

```
            y->r->p = 0;
        }
        y->r = x;
        x->p = y;
        x->pp = 0;
        update(y);
        splay(x);
    }
    return last;
}

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

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

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

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

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

void make_root(Node *x){
    access(x);
    x->flip = true;
```

```
176     swap(x->l, x->r);
177
178 }
179
180 class LinkCut
181 {
182     Node *x;
183
184 public:
185     LinkCut(int n)
186     {
187         x = new Node[n+5];
188         for(int i = 1; i <= n; i++)
189         {
190             x[i].label = i;
191             update(&x[i]);
192         }
193     }
194
195     virtual ~LinkCut()
196     {
197         delete[] x;
198     }
199
200     void link(int u, int v)
201     {
202         ::make_root(&x[u]);
203         ::link(&x[u], &x[v]);
204     }
205
206     void cut(int u, int v)
207     {
208         if(depth(u) > depth(v)) ::cut(&x[u]);
209         else ::cut(&x[v]);
210     }
211
212     int root(int u)
213     {
214         return ::root(&x[u])->label;
215     }
216
217     int depth(int u)
218     {
219         return ::depth(&x[u]);
220     }
221
222     int lca(int u, int v)
223     {
224         return ::lca(&x[u], &x[v])->label;
225     }
226 }*lctree;
```

```

char str[11];

int main(){
    //freopen("in.txt", "r", stdin);
    int n, m, u, v;

    scanf("%d %d", &n, &m);
    lctree = new LinkCut(n);

    while(m--){
        scanf("%s", str);
        if(str[0] == 'a'){
            scanf("%d %d", &u, &v);
            lctree -> link(u, v);
        }
        else if(str[0] == 'r'){
            scanf("%d %d", &u, &v);
            lctree -> cut(u, v);
        }
        else{
            scanf("%d %d", &u, &v);
            if(lctree->root(u) == lctree->root(v)) puts("YES");
            else puts("NO");
        }
    }
    return 0;
}

///11111
```

1.10 PBDS

```
#include <ext/pb_ds/assoc_container.hpp> // Common file
#include <ext/pb_ds/tree_policy.hpp> // Including
    tree_order_statistics_node_update

using namespace __gnu_pbds;

int main()
{
    /// Ordered Set
    typedef tree< int, null_type, less<int>, rb_tree_tag,
            tree_order_statistics_node_update> ordered_set;

    ordered_set X;
    X.insert(1);
    cout<<*X.find_by_order(1)<<endl; // 2
    cout<<X.order_of_key(-5)<<endl;  // 0


    /// Ordered Multiset. Notice the less_equal<int> parameter.
```

```
    typedef tree<int, null_type, less_equal<int>, rb_tree_tag,
            tree_order_statistics_node_update> ordered_multiset;

    ordered_multiset x;

    x.insert(0);
    x.insert(1);
    x.insert(1);
    x.insert(2);

    x.erase(x.find_by_order(x.order_of_key(0))); /// erasing is tricky

    cout<<*x.find_by_order(0)<<endl;

    /// Pair <int,int> Ordered Set
    typedef tree<pair<int,int>, null_type, less_equal<pair<int,int>>,
       rb_tree_tag,
            tree_order_statistics_node_update> ordered_pair;

    return 0;
}
```

1.11 Persistent treap

```
void persistent_split(Treap *t, Treap *&l, Treap *&r, LL key)
{
    if(!t) l = r = NULL;
    else if(t->val <= key) {
        *l = *t;
        l->r = new Treap(0);
        persistent_split(t->r, l->r, r, key);
        upd_sz(l);
    }
    else {
        *r = *t;
        r->l = new Treap(0);
        persistent_split(t->l, l, r->l, key);
        upd_sz(r);
    }
}


void persistent_merge(Treap *&t, Treap *l, Treap *r){
    if(!l || !r) { t = l ? l : r; return ;}
    if(l->prior > r->prior){
        *t = *l;
        t->r = new Treap(0);
        persistent_merge(t->r, l->r, r);
    }
    else{
        *t = *r;
        t->l = new Treap(0);
```

```
        persistent_merge(t->l, l, r->l);
    }
    upd_sz(t);
}

void persistent_insert(Treap *&t, Treap *old, LL key, LL x = 0){
    t = new Treap(0);
    Treap *l = new Treap(0), *r = new Treap(0), *m = new Treap(0);
    persistent_split(old, l, r, key);
    persistent_merge(m, l, new Treap(key));
    persistent_merge(t, m, r);
}


void persistent_erase(Treap *&t, Treap *old, LL key)
{
    t = new Treap(0);
    if(!old) {t = 0; return;}
    Treap *l = new Treap(0), *m1 = new Treap(0), *m2 = new Treap(0), *r = new
        Treap(0);
    persistent_split(old, l, r, key);
    persistent_split(l, m1, m2, key - 1);
    persistent_merge(t, m1, r);
}
```

1.12 Range update range query BIT

```

// Range_update_Range_query_BIT
// An array, suppose arr[MAX]
// 1 based indexing
LL BIT_ADD[MAX+10];
LL BIT_SUB[MAX+10];
int mxval;

void init()
{
    mem(BIT_ADD, 0);
    mem(BIT_SUB, 0);
}

void update(LL BIT[], int idx, LL val) //single point update, arr[idx] = val
{
    while(idx <= mxval)
        BIT[idx] += val, idx += idx&-idx;
}

LL query(LL BIT[], int idx) // single point query, cumulative sum from arr[1]
    to arr[idx]
{
    LL ret = 0;
    while(idx)
```

```
        ret += BIT[idx], idx -= idx&-idx;
    return ret;
}


LL range_query(int L, int R) // cumulative sum from arr[L] to arr[R]
{
    LL ret = (R*query(BIT_ADD, R) - (L-1)*query(BIT_ADD, L-1)) - (query(
        BIT_SUB, R) - query(BIT_SUB, L-1)) ;
    return ret;
}

void range_update(int L, int R, LL v) // For i = L to R, arr[i] += val
{
    update(BIT_ADD, L,v);
    update(BIT_ADD, R+1, -v);
    update(BIT_SUB, L, v*(L-1));
    update(BIT_SUB, R+1, -v*(R));
}
```

1.13 implicit seg tree

```

///Implicit Seg Tree
///For long range

struct node{
    int sum;
    node *l,*r;
    node():sum(0),l(0),r(0){}
};

void update(node *cur, int st, int ed, int pos, int val)
{
    if(st == ed)
    {
        cur->sum += val;
        return;
    }

    int mid = (st+ed)/2;
    if(!cur->l) cur->l = new node;
    if(!cur->r) cur->r = new node;

    if(pos <= mid) update(cur->l, st, mid, pos,val);
    else if(pos > mid) update(cur->r, mid+1,ed,pos,val);

    cur->sum = (cur->l)->sum + (cur->r)->sum;
}

int query(node *cur, int st, int ed, int i, int j)
{
```

```
    if(!cur) return 0;
    if(st == i && ed == j) return cur->sum;

    int mid=(st+ed)/2;
    if(j <= mid) return query(cur->l, st, mid, i,j);
    else if(i > mid) return query(cur->r, mid+1,ed,i,j);
    return query(cur->l,st,mid,i,mid) + query(cur->r,mid+1,ed,mid+1,j);
}

void rmv(node *cur)
{
    if(!cur) return;
    rmv(cur->l);
    rmv(cur->r);
    delete(cur);
}
```

1.14 lazy propagation template

```

struct lazy{
    LL lz;

    //Initialize here
    lazy(){
        lz = 0;
    }

    //Merge two lazies
    void impose(lazy ano){
        lz += ano.lz;
    }
};


bool notCleared(lazy L){ //Returns true if the lazy needs to be propagated
    return (bool)  L.lz;
}


struct node{
    //Insert the attributes here
    LL sum;
    lazy L;

    //The range it holds
    int l, r;

    //Initialize here
    node(){
        sum = l = r = 0;
    }
```

```
34
35      //Applies the given lazy on the attributes and imposes the lazy with its
            own lazy
36      void apply(lazy ano){
37          sum += ano.lz * (r - l + 1);
38          L.impose(ano);
39      }
40
41      void clearLazy(){
42          L.lz = 0;
43      }
44  } tree[MAX << 2];
45
46
47  void propagate(int idx)
48  {
49      int l = 2*idx, r = l + 1;
50
51      tree[l].apply(tree[idx].L);
52      tree[r].apply(tree[idx].L);
53
54      tree[idx].clearLazy();
55  }
56
57  void build_tree(int idx, int st, int ed)
58  {
59      tree[idx].l = st;
60      tree[idx].r = ed;
61      tree[idx].clearLazy();
62
63      if(st == ed){
64          //Base Case
65          tree[idx].sum = 0;
66          return;
67      }
68      int mid = (st+ed)/2, l = 2*idx, r = l + 1;
69      build_tree(l, st, mid);
70      build_tree(r, mid+1, ed);
71
72
73      //Merge the attributes here
74      tree[idx].sum = tree[l].sum + tree[r].sum;
75  }
76
77  void update(int idx, int st, int ed, int i, int j, lazy &curr)
78  {
79      if(st == i && ed == j)
80      {
81          tree[idx].apply(curr);
82          return;
83      }
```

```
    int mid = (st+ed)/2, l = 2*idx, r = l+1;
    if( notCleared(tree[idx].L) ) propagate(idx);

    if(j <= mid) update(l , st, mid, i, j, curr);
    else if(i > mid) update(r, mid+1, ed, i, j, curr);
    else update(l, st, mid, i, mid, curr), update(r, mid+1, ed, mid+1, j, curr
        );

    //Merge the attributes here
    tree[idx].sum = tree[l].sum + tree[r].sum;
}

LL query(int idx, int st, int ed, int i, int j)
{
    if(st == i && ed == j) return tree[idx].sum;
    int mid = (st+ed)/2, l = 2*idx, r = l + 1;

    if( notCleared(tree[idx].L) ) propagate(idx);

    if(j <= mid) return query(l, st, mid, i, j);
    if(i > mid) return query(r, mid+1, ed, i, j);
    return (query(l,st,mid,i,mid) + query(r,mid+1,ed,mid+1,j));
}
```

1.15 persistent segment tree

```

// Persistent segment tree
struct node{
    LL sum;
    node *l, *r;

    node(){
        l = r = 0;
        sum = 0;
    }
};

struct persistent_segment_tree{
    node *root[2*MAX+5];

    void build(node *cur, int st, int ed){
        if(st == ed) return;
        cur->l = new node;
        cur->r = new node;
        int mid = (st+ed)/2;
        build(cur->l, st, mid);
        build(cur->r, mid+1, ed);
    }

    persistent_segment_tree(int n = MAX+1){
```

```
        root[0] = new node;
        build(root[0], 1, n);
    }

    void update(node *cur, node *prv, int st, int ed, int pos, int val){
        if(st == ed){
            cur->sum = prv->sum + val;
            return;
        }

        int mid = (st+ed)/2;
        if(pos <= mid){
            cur->l = new node;
            cur->r = prv->r;
            update(cur->l, prv->l, st, mid, pos, val);
        }
        else{
            cur->l = prv->l;
            cur->r = new node;
            update(cur->r, prv->r, mid+1, ed, pos, val);
        }

        cur->sum = cur->l->sum + cur->r->sum;
    }

    LL query(node *cur, int st, int ed, int i, int j)
    {
        if(st == i && ed == j) return cur->sum;
        int mid = (st+ed)/2;
        if(j <= mid) return query(cur->l, st, mid, i, j);
        else if(i > mid) return query(cur->r, mid+1, ed, i, j);
        return query(cur->l, st, mid, i, mid) + query(cur->r, mid+1, ed, mid
            +1, j);
    }
};
```

1.16 rmq seg tree

```

// Range Minimum Query = Structre of Segment Tree [ Single Point update, range
    query ]
struct node{
    int min;
};

node tree[4*MAX];

void update_single_node(int idx, int val) // change here
{
    tree[idx].min= val;
    return;
}
```

```

void update(int idx, int st, int ed, int pos, int val) // single point update
{
    if(pos < st || pos > ed) return;
    if(st == ed)
    {
        update_single_node(idx, val);
        return;
    }
    int mid = (st+ed)/2, l = 2*idx, r = l+1;
    update(l,st,mid,pos,val);
    update(r,mid+1,ed,pos,val);
    tree[idx].min = min(tree[l].min, tree[r].min);
}

int query(int idx, int st, int ed, int i, int j) // Range Query
{
    if(st == i && ed == j) return tree[idx].min;
    int mid = (st+ed)/2, l =2*idx, r = l+1;
    if(j <= mid) return query(l,st,mid,i,j);
    if(i > mid) return query(r,mid+1,ed,i,j);
    return min(query(l,st,mid,i,mid), query(r,mid+1,ed,mid+1,j));
}
```

1.17 static data rmq

```

///RANGE_MINIMUM_QUEREY[STATIC DATA]
///Preprocessing O(n log n)
///Query O(1)
#define MAXLG   17
#define MAXN    30000

struct data{
    int val, idx;
    data(){;}
    data(int v, int i){
        val = v;
        idx = i;
    }
}rmq[MAXLG+5][MAXN+10];
data min(data a, data b) { return (a.val <= b.val)?a:b;}


void update(int in[], int n) // -> 1 based indexing [must be]
{
    int stp, i;
    for(stp = 0; (1<<stp) <= n; stp++)
        for(i = 1; i <= n; i++)
        {
            if(!stp) rmq[stp][i] = data(in[i], i);
            else if(i + (1<<stp) -1 > n) break;
```

```
            else rmq[stp][i] = min(rmq[stp-1][i], rmq[stp-1][i+(1<<(stp-1))]);
        }
}

data query(int l, int r)
{
    int mxs = sizeof(int) * 8 - 1 - __builtin_clz(r+1-l);
    return min(rmq[mxs][l], rmq[mxs][r-(1<<mxs)+1]);
}
```

1.18 static treap as basic BBST

```

//Static Basic BBST

int cur_rt;
int indx;
//Maintains Max Heap
struct Treap{
    int val, prior, cnt;
    int l, r;
    Treap(int v = 0): l(0), r(0), val(v), prior((rand() << 15) + rand()), cnt
        (1) {}
} rt[MAX+11]; // initialize the root if declared locally

void init_treap(){
    indx = cur_rt = 0;
    memset(rt, 0, sizeof(rt));
}

int sz(int t) {return (t == 0) ? 0:rt[t].cnt;}

void upd_sz(int t){
    if(t != 0) rt[t].cnt = 1 + sz(rt[t].l) + sz(rt[t].r);
}

void split(int t, int &l, int &r, int key)
{
    if(t == 0) l = r = 0;
    else if(rt[t].val <= key) {split(rt[t].r, rt[t].r, r, key); l = t;}
    else {split(rt[t].l, l, rt[t].l, key); r = t;}
    upd_sz(t);
}

void merge(int &t, int l, int r) //needed to erase something
{
    if(l == 0 || r == 0) t = (l == 0)? l:r;
    else if(rt[l].prior > rt[r].prior) {merge(rt[l].r, rt[l].r, r); t = l;}
    else {merge(rt[r].l, l, rt[r].l);t = r;}
    upd_sz(t);
}

```

```
//don't insert duplicate and allocate memory before insertion
//inserts just a node, not a Treap
void insert(int &t, Treap it)
{
    if(t == 0)
    {
        t = ++indx;
        rt[t] = it;
    }
    else if(it.prior > rt[t].prior){split(t, it.l, it.r, it.val); rt[++indx] =
        it; t = indx;}
    else if (it.val > rt[t].val) insert(rt[t].r, it);
    else insert(rt[t].l, it);
    upd_sz(t);
}

void erase(int &t, int key)
{
    if(t == 0) return;
    else if(rt[t].val == key) {int tmp = t; merge(t,rt[t].l, rt[t].r); rt[tmp]
        = 0;}
    else if(key > rt[t].val) erase(rt[t].r, key);
    else erase(rt[t].l, key);
    upd_sz(t);
}

bool find(int t, int key)
{
    if(t == 0) return false;
    if(rt[t].val == key) return true;
    return (key > rt[t].val)? find(rt[t].r, key):find(rt[t].l, key);
}

int find_kth(int cur, int k)
{
    if(sz(rt[cur].l) < k)
    {
        k -= sz(rt[cur].l);
        if(k == 1) return rt[cur].val;
        return find_kth(rt[cur].r, k-1);
    }
    return find_kth(rt[cur].l, k);
}
```

1.19 static treap as dynamic array

```

//Static Treap As Dynamic Array
const int inf = 1e9;
int indx, cur_rt;
struct Treap{
    int val, prior, cnt;
```

```
7       int l, r;
8       Treap(int v = inf): l(inf), r(inf), val(v), prior((rand() << 15) + rand())
            , cnt(1) {}
9   }Rt[MAX+11];
10
11  void init_treap(){
12      indx = -1;
13      cur_rt = inf;
14  }
15
16  int sz(int t) {return (t == inf) ? 0:Rt[t].cnt;}
17
18  void upd_sz(int t){
19      if(t != inf) Rt[t].cnt = 1 + sz(Rt[t].l) + sz(Rt[t].r);
20  }
21
22  void split(int t, int &l, int &r, int pos, int add = 0)
23  {
24      if(t == inf) return void(l = r = inf);
25      int cur_pos = add + sz(Rt[t].l) + 1;
26      if(cur_pos <= pos) split(Rt[t].r, Rt[t].r, r, pos, cur_pos), l = t;
27      else split(Rt[t].l, l ,Rt[t].l, pos, add), r = t;
28      upd_sz(t);
29  }
30
31
32  void merge(int &t, int l, int r) //needed to erase something
33  {
34      if(l == inf || r == inf) t = (l != inf)? l:r;
35      else if(Rt[l].prior > Rt[r].prior) {merge(Rt[l].r, Rt[l].r, r); t = l;}
36      else {merge(Rt[r].l, l, Rt[r].l);t = r;}
37      upd_sz(t);
38  }
39
40
41  void insert(int &t, int pos, int val)
42  {
43      int l, r;
44      Rt[++indx] = Treap(val);
45      split(t, l, r, pos-1);
46      merge(t, l, indx);
47      merge(t, t, r);
48  }
49
50
51  void erase(int &t, int pos)
52  {
53      int l, r, g;
54      split(t, l, r, pos-1);
55      split(r, g, r, 1);
56      merge(t, l, r);
```

```
}


int find_kth(int t, int k, int add = 0)
{
    assert(t != inf);
    int cur_pos = add + sz(Rt[t].l) + 1;
    if(cur_pos == k) return Rt[t]. val;
    if(cur_pos < k) return find_kth(Rt[t].r, k, cur_pos);
    return find_kth(Rt[t].l, k, add);
}
```

1.20 treap Basic BBST

```

//Maintains Max Heap
struct Treap{
    int val, prior, cnt;
    Treap *l, *r;
    Treap(int v): l(NULL), r(NULL), val(v), prior((rand() << 15) + rand()),
        cnt(1) {}
}; // initialize the root if declared locally

int sz(Treap *t) {return (t == NULL) ? 0:t->cnt;}

void upd_sz(Treap *t){
    if(t) t->cnt = 1 + sz(t->l) + sz(t->r);
}

void split(Treap *t, Treap *&l, Treap *&r, int key)
{
    if(!t) l = r = NULL;
    else if(t->val <= key) {split(t->r, t->r, r, key); l = t;}
    else {split(t->l, l, t->l, key); r = t;}
    upd_sz(t);
}

void merge(Treap *&t, Treap *l, Treap *r) //needed to erase something
{
    if(!l || !r) t = l? l:r;
    else if(l->prior > r->prior) {merge(l->r, l->r, r); t = l;}
    else {merge(r->l, l, r->l);t = r;}
    upd_sz(t);
}

//don't insert duplicate and allocate memory before insertion
//inserts just a node, not a Treap
void insert(Treap *&t, Treap *it)
{
    if(!t) t = it;
    else if(it->prior > t->prior){split(t, it->l, it->r, it->val);t = it;}
    else if (it->val > t->val) insert(t->r, it);
```

```
38      else insert(t->l, it);
39      upd_sz(t);
40  }
41
42  void erase(Treap *&t, int key)
43  {
44      if(!t) return;
45      else if(t->val == key) {Treap *temp = t; merge(t,t->l, t->r); delete(temp)
            ;}
46      else if(key > t->val) erase(t->r, key);
47      else erase(t->l, key);
48      upd_sz(t);
49  }
50
51  bool find(Treap *t, int key)
52  {
53      if(!t) return false;
54      if(t->val == key) return true;
55      return (key > t->val)? find(t->r, key):find(t->l, key);
56  }
57
58  void rmv(Treap *u)
59  {
60      if(!u) return;
61      rmv(u->l);
62      rmv(u->r);
63      delete(u);
64  }
65
66  void shift(Treap *&u, Treap *v)
67  {
68      if(!v) return;
69      insert(u, new Treap(v->val));
70      shift(u, v->l);
71      shift(u, v->r);
72  }
73
74  Treap* join(Treap *u, Treap *v)
75  {
76      if(sz(u) < sz(v)) swap(u,v);
77      shift(u,v);
78      rmv(v);
79      return u;
80  }
81
82  int find_kth(Treap *cur, int k)
83  {
84      if(sz(cur->l) < k)
85      {
86          k -= sz(cur->l);
87          if(k == 1) return cur->val;
```

```
88          return find_kth(cur->r, k-1);
89      }
90      return find_kth(cur->l, k);
91  }
92
93  int predecessor(Treap *cur, int key) //Greatest one smaller than key
94  {
95      if(!cur) return -inf;
96      if(cur->val >= key) return predecessor(cur->l, key);
97      return max(cur->val, predecessor(cur->r,key));
98  }
99
100 int successor(Treap *cur, int key) //Smallest one greater than key
101 {
102     if(!cur) return inf;
103     if(cur->val <= key) return successor(cur->r,key);
104     return min(cur->val, successor(cur->l, key));
105 }
106
107 int cntLessOrE(Treap *cur, int key)
108 {
109     if(!cur) return 0;
110
111     if(cur->val <= key) return 1 + sz(cur->l) + cntLessOrE(cur->r, key);
112     return cntLessOrE(cur->l, key);
113 }
114
115 //Build Treap in O(N)
116 void build_treap(Treap *&t, int *arr, int st, int ed, int cnt = 1e9)
117 {
118     if(st > ed) return void(t == NULL);
119
120     if(st == ed)
121     {
122         t = new Treap(arr[st], cnt);
123         return;
124     }
125
126     int mid = (st+ed) >> 1;
127     t = new Treap(arr[mid], cnt);
128     cnt >>= 1;
129
130     build_treap(t->l, arr, st, mid-1, cnt);
131     build_treap(t->r, arr, mid+1,ed, cnt);
132     upd_sz(t);
133 }
134
135 void persistent_split(Treap *t, Treap *&l, Treap *&r, int key)
136 {
137     if(!t) l = r = NULL;
138     else if(t->val <= key) {
```

```
        *l = *t;
        l->r = new Treap(0);
        persistent_split(t->r, l->r, r, key);
        upd_sz(l);
    }
    else {
        *r = *t;
        r->l = new Treap(0);
        persistent_split(t->l, l, r->l, key);
        upd_sz(r);
    }
}


void persistent_merge(Treap *&t, Treap *l, Treap *r){
    if(!l || !r) { t = l ? l : r; return ;}
    if(l->prior > r->prior){
        *t = *l;
        t->r = new Treap(0);
        persistent_merge(t->r, l->r, r);
    }
    else{
        *t = *r;
        t->l = new Treap(0);
        persistent_merge(t->l, l, r->l);
    }
    upd_sz(t);
}

void persistent_insert(Treap *&t, Treap *old, int key){
    t = new Treap(0);
    Treap *l = new Treap(0), *r = new Treap(0), *m = new Treap(0);
    persistent_split(old, l, r, key);
    persistent_merge(m, l, new Treap(key));
    persistent_merge(t, m, r);
}
```

1.21 treap Dynamic Array

```

// Treap- Dynamic Array
struct Treap{
    int val, prior, cnt;
    Treap *l, *r;
    Treap(int v): l(NULL), r(NULL), val(v), prior((rand() << 15) + rand()),
       cnt(1) {}
}; // initialize the root if declared locally

int sz(Treap *t) {return (t == NULL) ? 0:t->cnt;}

void upd_sz(Treap *t){
    if(t) t->cnt = 1 + sz(t->l) + sz(t->r);
```

```
}

void split(Treap *t, Treap *&l, Treap *&r, int pos, int add = 0)
{
    if(!t) return void(l = r = NULL);
    int cur_pos = add + sz(t->l) + 1;
    if(cur_pos <= pos) split(t->r, t->r, r, pos, cur_pos), l = t;
    else split(t->l, l ,t->l, pos, add), r = t;
    upd_sz(t);
}

void merge(Treap *&t, Treap *l, Treap *r)
{
    if(!l || !r) t = l? l:r;
    else if(l->prior > r->prior) {merge(l->r, l->r, r); t = l;}
    else {merge(r->l, l, r->l);t = r;}
    upd_sz(t);
}

void insert(Treap *&t, int pos, int val)
{
    Treap *l, *r, *cur = new Treap(val);
    split(t, l, r, pos-1);
    merge(t, l, cur);
    merge(t, t, r);
}

void erase(Treap *&t, int pos)
{
    Treap *l, *r, *g;
    split(t, l, r, pos-1);
    split(r, g, r, 1);
    merge(t, l, r);
}

int find_kth(Treap *t, int k, int add = 0)
{
    assert(t);
    int cur_pos = add + sz(t->l) + 1;
    if(cur_pos == k) return t-> val;
    if(cur_pos < k) return find_kth(t->r, k, cur_pos);
    return find_kth(t->l, k, add);
}

void rmv(Treap *u)
{
    if(!u) return;
    rmv(u->l);
    rmv(u->r);
    delete(u);
}
```

2 Dynamic Programming

2.1 Convex Hull Trick

```

struct Line{
        LL m,c;
        Line(LL _m = 0, LL _c = 0):m(_m), c(_c){};
};

struct ConvexHullTrick{ //works with long long integers.

    vector<Line> Q; //Fast -> Slow -> Slower -> Slowest
    bool minFlag;

    ConvexHullTrick(bool flg = false):minFlag(flg){};

     LL getX(Line u, Line v){ // Fast vrs Slow *ORDER MATTERS*
        LL difC = v.c - u.c, difM = u.m - v.m;
        if(difC % difM == 0) return difC/difM;
        if(difC < 0) return difC/difM;
        return difC/difM + 1;
    }


    bool isBad(Line L1, Line L2, Line L3)
    {
        if(minFlag == false) return (L3.c - L1.c) / (long double) (L1.m - L3.m) > (L2.c-L1.c) / (long double) (L1.m - L2.m);
        else return (L3.c - L1.c) / (long double) (L1.m - L3.m) < (L2.c-L1.c) / (long double) (L1.m - L2.m);
    }

    void addLine(Line L){ //Has to be slower than then the slowest in the Q
        while(Q.empty() == false)
        {
            if(Q.back().m < L.m) __builtin_trap();
            else if(minFlag == false && Q.back().m == L.m && L.c > Q.back().c) Q.pop_back();
            else if(minFlag == true && Q.back().m == L.m && L.c < Q.back().c) Q.pop_back();
            else if(Q.back().m == L.m) return;
            else if(Q.size() <= 1) break;
            else if(isBad(Q[Q.size()-2], Q.back(), L)) Q.pop_back();
            else break;
        }
        Q.push_back(L);
    }

    LL query(LL pos){
        int lo = 0, hi = (int) Q.size() - 1, n = hi, mid;
        LL L, R;
```

```
45
46          while(true)
47          {
48              mid = (lo+hi)/2;
49              if(minFlag)
50              {
51                  if(mid == 0) L = -5e18;
52                  else L = getX(Q[mid-1], Q[mid]);
53
54                  if(mid == n) R = 5e18;
55                  else R = getX(Q[mid], Q[mid+1]);
56
57                  if(L <= pos && pos < R) return Q[mid].m * pos + Q[mid].c;
58                  if(pos < L) hi = mid-1;
59                  else lo = mid+1;
60              }
61
62              else
63              {
64                  if(mid == n) L = -5e18;
65                  else L = getX(Q[mid], Q[mid+1]);
66
67                  if(mid == 0) R = 5e18;
68                  else R = getX(Q[mid-1], Q[mid]);
69
70                  if(L <= pos && pos < R) return Q[mid].m * pos + Q[mid].c;
71                  if(pos < L) lo = mid+1;
72                  else hi = mid-1;
73              }
74          }
75      }
76  };
```

2.2 LDS nlogk

```
1
2   // lds - O(nlogk)
3   int fast_lds(int *arr, int N) //Calculates lds_len of arr[1]...arr[N] and
        returns lds
4   {
5       int ret = 1;
6       vector<int> sq;
7       int i, lo, hi, mid;
8
9       sq.pb(arr[1]);
10      lds_len[1] = 1;
11
12      for(i = 2; i <= N; i++)
13      {
14          lo = 0;
15          hi = sq.size()-1;
16
```

```
        if(num[i] >= sq[lo]) sq[lo] = num[i], lds_len[i] = 1;
        else if(num[i] <  sq[hi]) sq.pb(num[i]), lds_len[i] = hi+2;
        else
        {
            while(lo < hi)
            {
                mid = lo+ ((hi-lo+1) >> 1);
                if(sq[mid] > arr[i]) lo = mid;
                else hi = mid-1;
            }
            sq[lo+1] = arr[i];
            lds_len[i] = lo+2;
        }

        ret = max(ret, lds_len[i]);
    }
    return ret;
}
```

2.3 LIS nlogk

```

// LIS - O(nlogk)
int fast_lis(int *arr, int N) //Calculates lis_len of arr[1]...arr[N] and
    returns lis
{
    int ret = 1;
    vector<int> sq;
    int i, lo, hi, mid;

    sq.pb(arr[1]);
    lis_len[1] = 1;

    for(i = 2; i <= N; i++)
    {
        lo = 0;
        hi = sq.size()-1;

        if(num[i] <= sq[lo]) sq[lo] = num[i], lis_len[i] = 1;
        else if(num[i] >  sq[hi]) sq.pb(num[i]), lis_len[i] = hi+2;
        else
        {
            while(lo < hi)
            {
                mid = lo+ ((hi-lo+1) >> 1);
                if(sq[mid] < arr[i]) lo = mid;
                else hi = mid-1;
            }
            sq[lo+1] = arr[i];
            lis_len[i] = lo+2;
        }

```

```
        ret = max(ret, lis_len[i]);
    }
    return ret;
}
```

2.4 SHORT LIS

```
/*
Finds only LIS. LDS can be found by simply multiplying the whole input array
   with -1.
For Longest Non-Decreasing sequence, simply use upper_bound().
Complexity: NlogK
*/
struct LIS {
    int bbb[NSIZE+10];

    int calculateLIS ( int arr[], int lisVal[], int n ) {
        FOR(i,0,n) {
            bbb[i] = inf;
        }
        bbb[0] = -inf;

        int mx = 0;
        FOR(i,0,n-1) {
            int v = arr[i];
            int pos = lower_bound ( bbb, bbb + mx + 1, v ) - bbb;
            lisVal[i] = pos;
            bbb[pos] = v;
            mx = MAX(mx,pos);
        }

        return mx;
    }
}lis;
```

2.5 SOS DP

```

/* SOS DP
    Size of the dp table has to be quite large
    Don't forget to memset the dp array with 0 from main() function
*/

void sos_dp(int *arr, LL dp[], int n){

    int k = 1, i, pos, mask;

    while((1 << k) <= n) k++;
    for(i = 0; i <= n; i++)
        dp[i] = arr[i];

    for(pos = 0; pos < k; pos++)
```

```
        for(mask = 0; mask < (1 << k); mask++)
            if(mask & (1 << pos))
                dp[mask] += dp[mask ^ (1 << pos)];
}
```

2.6 dynamic cht

```

const LL is_query = -(1LL<<62);
struct Line {
    LL m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        LL x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> { // will maintain upper hull for maximum
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;

        return (__int128) (x->b - y->b) * (z->m - y->m) >= (__int128)(y->b - z->b) * (y->m - x->m);
    }
    void insert_line(LL m, LL b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    LL eval(LL x) {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

2.7 josephus

```

//Josephus [ 0 indexed ] O (n)
int josephus(int n, int k)
```

```
{
    if(n == 1) return 0;
    return (josephus(n-1, k) + k)%n;
}
```

2.8 tree dp

```

inline int maxLoad(int idx){
    if(idx == -1) return 0;
    if(ml[idx] != -1) return ml[idx];
    return ml[idx] = sz[idx] + maxLoad(sibling[idx]);
}

int calc(int idx, int k, bool dflag)
{
    if(k == 0) return 0;
    if(idx == -1) return inf;
    if(k > maxLoad(idx)) return inf;
    if(dp[dflag][k][idx] != -1) return dp[dflag][k][idx];

    int with = inf, without = inf, current;
    int x = min(maxLoad(fc[idx]), k), y = min(maxLoad(fc[idx]), k - 1);
    for(int i = max(0, k - maxLoad(sibling[idx])); i <= x; i++) without = min(
        without, calc(fc[idx], i, false) + calc(sibling[idx], k - i, dflag));
    for(int i = max(0, k - 1 - maxLoad(sibling[idx])); i <= y; i++) with = min
        (with, calc(fc[idx], i, dflag) + calc(sibling[idx], k - 1 - i, dflag))
        ;

    with += c[idx];
    if(dflag) with -= d[idx];
    return dp[dflag][k][idx] = min(inf , min(with, without));
}
```

3 Geometry

3.1 3D Geometry

```

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

#define pi      acos(-1.00)
#define eps     1e-11
#define D(x)    cout << #x " = " << (x) << endl

bool eq(double a, double b) { return fabs( a - b ) < eps; } //two numbers are
    equal

/*
Good to know:
-> Cross product is distributive over addition: a x (b + c) = a x b + a x c
```

```
-> Cross product is anticommutative  a x b = -(b x a)
-> Cross product is not associative a x ( b x c ) != (a x b) x c
*/

struct point{
    double x, y, z;
    point(){}
    point(double xx, double yy, double zz) {x = xx, y = yy, z = zz;} // NEVER USE xx = 0 or yy = 0 HERE

    void takeInput(){
        cin >> x >> y >> z;
    }

} origin = point(0, 0, 0); //OK

point operator+(const point &u, const point &v) {return point(u.x + v.x, u.y + v.y, u.z + v.z);} //OK
point operator-(const point &u, const point &v) {return point(u.x - v.x, u.y - v.y, u.z - v.z);} //OK
point operator*(const point &u, double v) {return point(u.x*v, u.y*v, u.z*v);} //OK
point operator*(double v, const point &u) {return point(u.x*v, u.y*v, u.z*v);} //OK

point operator/(const point &u, double v) {assert(abs(v) > eps); return point(u.x/v, u.y/v, u.z/v);}
bool operator != (const point &u, const point &v) {return !(eq(u.x, v.x) && eq(u.y, v.y) && eq(u.z, v.z));}

ostream &operator <<(ostream &os, const point &p) {
  os << "(" << p.x << "," << p.y << "," << p.z << ")";
}

double norm(point u){return sqrt(u.x * u.x + u.y * u.y + u.z * u.z);} //OK
double dotp(point u, point v) {return u.x * v.x + u.y * v.y + u.z * v.z;} //OK
point crsp(point u, point v) {return point(u.y*v.z-u.z*v.y, u.z*v.x-u.x*v.z, u.x*v.y-u.y*v.x);} //OK

struct plane{
    double a, b, c, d; //ax + by + cz + d = 0
    plane(){;}
    plane(point p1, point p2, point p3){
        point vtr = crsp(p2 - p1, p3 - p1);
        if(norm(vtr) < eps) {assert(false);} // doesn't define a plance
        a = vtr.x;
        b = vtr.y;
        c = vtr.z;
        d = -p1.x*vtr.x -p1.y*vtr.y - p1.z * vtr.z;
    }
};
```

```
57
58  double smlr_angle(point l, point m, point r)
59  {
60      double d = dotp(l - m, r - m);
61      return acos(d / (norm(l-m) * norm(r-m)));
62  }
63
64  point unit_vector(point u) { return u / norm(u); } //OK
65  point projection(point p, point st, point ed) { return dotp(ed - st, p - st) /
        norm(ed - st) * unit_vector(ed - st) + st;} //OK
66  point extend(point st, point ed, double len) { return ed + unit_vector(ed-st)
        * len;} //OK
67
68  point rtt(point axis, point p, double theta){
69      axis = unit_vector(axis);
70      return p * cos(theta) +  sin(theta) * crsp(axis, p) + axis * (1-cos(theta)
            ) * dotp(axis, p);
71  } //OK
72
73  point segmentProjection(point p, point st, point ed)
74  {
75      double d = dotp(p - st, ed - st) / norm(ed - st);
76      if(d < 0) return st;
77      if(d > norm(ed - st) + eps) return ed;
78      return st + unit_vector(ed - st) * d;
79  } //OK
80
81  double distancePointSegment(point p, point st, point ed) {return norm(p -
        segmentProjection(p, st, ed)); } //OK
82  double distancePointLine( point P, point st, point ed) { return norm(
        projection(P, st, ed) - P ); } //OK
83
84  double pointPlaneDistance(plane P, point q){
85      return fabs(P.a * q.x + P.b * q.y + P.c * q.z + P.d) / sqrt(P.a * P.a + P.
            b * P.b + P.c * P.c); //OK
86  }
87  double pointPlaneDistance(point p1, point p2, point p3, point q){ return
        pointPlaneDistance(plane(p1,p2,p3), q); } //OK
88
89  point reflection(point p, point st, point ed){
90      point proj = projection(p, st, ed);
91      if(p != proj) return extend(p, proj, norm(p - proj));
92      return proj;
93  } //OK
94
95  bool coplanar(point p1, point p2, point p3, point q)
96  {
97      p2 = p2-p1, p3 = p3-p1, q = q-p1;
98      if( fabs( dotp(q, crsp(p2, p3)) ) < eps ) return true;
99      return false;
100 }
```

```

int linePlaneIntersection(point u, point v, point l, point m, point r, point &x){
    /*
        -> l, m, r defines the plane
        -> u, v defines the line
        -> returns 0 when does not intersect
        -> returns 1 when there exists one unique common point
        -> returns -1 when there exists infinite number of common point
    */

    assert(l != m && m != r && l != r && u != v);
    if(coplanar(l, m, r, u) && coplanar(l, m, r, v)) return -1;
    l = l - m;
    r = r - m;
    u = u - m;
    v = v - m;

    point C = crsp(l, r);
    double denom = dotp(v - u, C);
    if(fabs(denom) < eps) return 0;

    double alpha = -dotp(C, u) / denom;
    x = u + (v - u) * alpha + m;

    return 1;
}

double angle(point u, point v) { return acos(dotp(u, v) / (norm(u) * norm(v)));}


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

3.2 Adaptive Simpson

```

// Template Credit: dragoon

#define z_slice_eps 1e-5
#define SIMPSON_EPS 1e-9
#define SIMPSON_TERMINAL_EPS 1e-12

double F(double x) {
    return x;
}

double single_simpson(double miny, double maxy) {
    return (maxy - miny) / 6 * (F(miny) + 4 * F((miny + maxy) / 2.) + F(maxy))
```

```
        ;
}

double adaptive_simpson(double miny, double maxy, double c, double eps = SIMPSON_EPS) {
    if(maxy - miny < SIMPSON_TERMINAL_EPS) return 0;

    double midy = (miny + maxy) / 2;
    double a = single_simpson(miny, midy);
    double b = single_simpson(midy, maxy);

    if(fabs(a + b - c) < 15 * eps) return a + b + (a + b - c) / 15.0;

    return adaptive_simpson(miny, midy, a, eps / 2.) + adaptive_simpson(midy, maxy, b, eps / 2.);
}

double simpson(double minz, double maxz)
{
    double ans, last, z, temp;

    ans = 0;
    last = F(minz);
    for(z = minz; z<=maxz; z+=z_slice_eps)
    {
        if(z>(minz+maxz)/2)
            z = z;

        temp = F(z+z_slice_eps);
        ans += last + 4*F(z+z_slice_eps/2) + temp;
        last = temp;
    }

    ans *= z_slice_eps/6;

    return ans;
}

double Integrate(double x_st, double x_ed) { return adaptive_simpson(x_st, x_ed, single_simpson(x_st, x_ed));}
```

3.3 All pair closest point

```

#include<bits/stdc++.h>
using namespace std;
#define D(x)         cout<<#x " = "<<(x)<<endl
#define un(x)        x.erase(unique(x.begin(),x.end()), x.end())
#define sf(n)        scanf("%d", &n)
#define sff(a,b)     scanf("%d %d", &a, &b)
#define sfff(a,b,c)  scanf("%d %d %d", &a, &b, &c)
#define pb           push_back
```

```
10 #define mp          make_pair
11 #define xx          first
12 #define yy          second
13 #define hp          (LL) 999983
14 #define MAX         100005
15 #define eps         1e-9
16 #define pi          acos(-1.00)
17 typedef long long int LL;
18 typedef pair<int,int> pii;
19 typedef vector<pii> vpii;
20
21 template<class T> void checkmin(T &a,T b){if(b<a) a=b;}
22 template<class T> void checkmax(T &a,T b){if(b>a) a=b;}
23
24 const int maxn = 100005;
25 int tx[maxn];
26 int ty[maxn];
27 bool divX[maxn];
28 pii key[maxn];
29
30 bool cmpX(const pii &a, const pii &b) {
31     return a.first < b.first;
32 }
33
34 bool cmpY(const pii &a, const pii &b) {
35     return a.second < b.second;
36 }
37
38 void buildTree(int left, int right, pii points[]) {
39     if (left >= right)
40         return;
41     int mid = (left + right) >> 1;
42
43     //sort(points + left, points + right + 1, divX ? cmpX : cmpY);
44     int minx = INT_MAX;
45     int maxx = INT_MIN;
46     int miny = INT_MAX;
47     int maxy = INT_MIN;
48     for (int i = left; i < right; i++) {
49         checkmin(minx, points[i].first);
50         checkmax(maxx, points[i].first);
51         checkmin(miny, points[i].second);
52         checkmax(maxy, points[i].second);
53     }
54     divX[mid] = (maxx - minx) >= (maxy - miny);
55     nth_element(points + left, points + mid, points + right, divX[mid] ? cmpX
       : cmpY);
56
57     tx[mid] = points[mid].first;
58     ty[mid] = points[mid].second;
59
```

```
60      if (left + 1 == right)
61          return;
62      buildTree(left, mid, points);
63      buildTree(mid + 1, right, points);
64  }
65
66  long long closestDist;
67  int closestNode;
68
69  void findNearestNeighbour(int left, int right, int x, int y) {
70      if (left >= right)
71          return;
72      int mid = (left + right) >> 1;
73      int dx = x - tx[mid];
74      int dy = y - ty[mid];
75      long long d = dx * (long long) dx + dy * (long long) dy;
76      if (closestDist > d && d) {
77          closestDist = d;
78          closestNode = mid;
79      }
80      if (left + 1 == right)
81          return;
82
83      int delta = divX[mid] ? dx : dy;
84      long long delta2 = delta * (long long) delta;
85      int l1 = left;
86      int r1 = mid;
87      int l2 = mid + 1;
88      int r2 = right;
89      if (delta > 0)
90          swap(l1, l2), swap(r1, r2);
91
92      findNearestNeighbour(l1, r1, x, y);
93      if (delta2 < closestDist)
94          findNearestNeighbour(l2, r2, x, y);
95  }
96
97  void buildTree(int n, pii points[])
98  {
99      for(int i = 0; i < n; i++)
100         key[i] = points[i];
101
102     buildTree(0, n, key);
103 }
104
105 int findNearestNeighbour(int n, int x, int y) {
106     closestDist = LLONG_MAX;
107     findNearestNeighbour(0, n, x, y);
108     return closestNode;
109 }
110
```

```
pii pnt[MAX+5];

int main()
{
    //freopen("in.txt", "r", stdin);

    int i, j, k, t, cs;
    int n;

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

        buildTree(n, pnt);
        for(i = 0; i < n; i++)
        {
            findNearestNeighbour(n, pnt[i].xx, pnt[i].yy);
            printf("%lld\n", closestDist);
        }
    }

    return 0;
}
```

3.4 AreaOfSubPolygon

```
//Area of Sub-Polygon [st ... ed]

vector< pii > polygon;
LL dp[MAX+5];

LL F(int ed)
{
    if(ed < 0) return 0;
    if(dp[ed] != -1) return dp[ed];

    int nxt = (ed + 1) % polygon.size();
    return dp[ed] = (LL) polygon[ed].xx * polygon[nxt].yy - (LL) polygon[ed].
       yy * polygon[nxt].xx + F(ed - 1);
}

LL compute(int st, int ed)
{
    LL ret = (LL) polygon[ed].xx * polygon[st].yy - (LL)polygon[ed].yy *
       polygon[st].xx;
    if(st <= ed) return abs( ret + F(ed - 1) - F(st - 1));
    return abs( ret + F((int) polygon.size() - 1) - (F(st - 1) - F(ed - 1)));
}
```

3.5 Convex Hull

```
1
2
3  #define xx      first
4  #define yy      second
5  typedef long long int LL;
6  typedef pair<int,int> pii;
7
8  #define CW      -1
9  #define ACW      1
10
11 int direction(pii st, pii ed, pii q){
12     LL xp = (LL) (ed.xx - st.xx) * (q.yy - ed.yy) - (LL) (ed.yy - st.yy) * (q.
           xx - ed.xx);
13     if(!xp) return 0;
14     if(xp > 0) return ACW;
15     return CW;
16 }
17
18 /*
19     Minimized border points, works in degenerate case
20     To maximize border points
21         - change != to == and swap(CW,ACW)
22         - Be careful about p[0], p[back] line
23 */
24 int convex_hull(vector <pii> p, vector<pii> &h)
25 {
26     vector<pii> up, dwn; // constructs upper hull in clockwise order, lower
           hull in anti-clockwise order
27     h.clear();
28     sort(p.begin(), p.end());
29     up.push_back(p[0]);
30     dwn.push_back(p[0]);
31
32     for(int i = 1; i < (int) p.size(); i++){
33         if(direction(p[0], p.back(), p[i]) != CW){
34             while(up.size() >= 2 && direction(up[up.size() - 2], up.back(), p[
                   i]) != CW) up.pop_back();
35             up.push_back(p[i]);
36         }
37
38         if(direction(p[0], p.back(), p[i]) != ACW){
39             while(dwn.size() >= 2 && direction(dwn[dwn.size() - 2], dwn.back()
                   , p[i]) != ACW) dwn.pop_back();
40             dwn.push_back(p[i]);
41         }
42     }
43
44     h = dwn;
45     for(int i = (int) up.size() - 2; i >= 1; i--) h.push_back(up[i]);
```

```
    return h.size();
}
```

3.6 Dynamic Convex Hull

```
// Dynamic Convex Hull
struct point{
    LL x, y;
    point(){}
    point(int xx, int yy){
        x = xx;
        y = yy;
    }
};

bool operator < (const point &u, const point &v){
    if(u.x == v.x) return u.y < v.y;
    return u.x < v.x;
}

LL signedArea(point a, point b, point c)
{
    LL ret = (LL) a.x* (b.y - c.y) + (LL) b.x*(c.y - a.y) + (LL) c.x*(a.y - b.
        y);
    if(ret < 0) return -1;
    if(ret > 0) return +1;
    return 0;
}

LL triArea(point a, point b, point c) { return abs((LL) a.x* (b.y - c.y) + (LL
    ) b.x*(c.y - a.y) + (LL) c.x*(a.y - b.y)); }

struct Dynamic_Convex_Hull{

    deque< point > hull;
    int n;
    LL Area;

    Dynamic_Convex_Hull(){
        Area = 0;
        n = 0;
        hull.clear();
    }

    bool brute(point p)
    {
        if(n < 3){
            n++;
            hull.push_back(p);
            if(n == 3 && signedArea(hull[0], hull[1], hull[2]) < 0) swap(hull
                [0], hull[1]);
            if(n == 3) Area = triArea(hull[0], hull[1], hull[2]);
```

```
            return true;
        }
        return false;
    }

    void addPoint(point p)
    {
        while(true){
            if(brute(p)) return;
            if(signedArea(p, hull.back(), hull[hull.size() - 2]) > 0){
                hull.push_front(hull.back());
                hull.pop_back();
            }
            else  break;
        } //Went down to the lower Tangent


        point previous = hull.back();
        while(true){
            if(brute(p)) return;
            if(signedArea(p, hull[0], hull[1]) < 0)
            {
                Area += triArea(p, hull[0], previous);
                previous = hull.front();
                n--, hull.pop_front();
            }
            else break;
        }
        Area += triArea(p, hull[0], previous);

        n++;
        hull.push_back(p);
    }
}solver;
```

3.7 Integer Line

```

//Geo_Line
struct Line{
    LL a, b, c;
    Line(){}
    Line(LL x, LL y, LL z){
        a = x;
        b = y;
        c = z;
    }
    Line(pii p1, pii p2) //ax+by+c=0
    {
        *this = Line(p1.yy-p2.yy, p2.xx-p1.xx, -(LL)p1.xx *(p1.yy-p2.yy) + (LL
            )p1.yy * (p1.xx-p2.xx));
        LL g = __gcd(__gcd(a,b),c);
```

```

        assert(g);
        if(a < 0 || (!a && b < 0) || (!(a || b) && c < 0))g = -abs(g);
        else if(g < 0) g = -g;

        a /= g; b /= g; c /= g;
    }
};
```

3.8 antipodal

```

vector< pii > antipodal(vector<point> polygon){
    /*
        No 3 points can be co-linear
        Has to be in anti-clockwise order
        Can't be degenerate
        ret[i] = range of vertices antipodal to the i-th vertex
    */
    vector<pii> ret;
    vector<point> nxt;
    int q_lo, q_hi, n = polygon.size(), i, mxCnt;
    for(i = 0; i < n; i++) nxt.push_back(polygon[(i+1) % n]);

    q_lo = 0;
    while(area(polygon[n-1], nxt[n-1], nxt[q_lo]) > area(polygon[n-1], nxt[n
        -1], polygon[q_lo]))
        q_lo = (q_lo + 1) % n;

    for(i = 0; i < n; i++)
    {
        if(q_lo == i) q_lo = (q_lo + 1) % n;

        mxCnt = 0;
        q_hi = q_lo;
        while(true){
            double h1 = area(polygon[i], nxt[i], nxt[q_hi]), h2 = area(polygon
                [i], nxt[i], polygon[q_hi]);
            if(h1 > h2 + eps) q_hi = (q_hi + 1) % n, mxCnt = 0;
            else if(eq(h1, h2)) q_hi = (q_hi + 1) % n, mxCnt++;
            else break;
        }
        ret.push_back(pii(q_lo, q_hi));
        q_lo = (q_hi - (mxCnt)) % n;
        if(q_lo < 0) q_lo += n;
    }
    return ret;
}
```

3.9 circle rectangle intersection

```

```

```
2  //Circle Rectangle Intersection
3  double areaArc( double r, double x1, double y1 )
4  {
5      double x2 = sqrt( r*r - y1*y1 );
6      double y2 = sqrt( r*r - x1*x1 );
7      double theta = acos( ( 2*r*r - (x2-x1)*(x2-x1) - (y2-y1)*(y2-y1) ) / ( 2*r
           *r ) );
8      return (theta*r*r - y1 * (x2 - x1) - x1 * (y2 - y1)) / 2;
9  }
10
11 double circleRectangleIntersection( int r, int x1, int y1, int x2, int y2 )
12 {
13     if( x1 < 0 && x2 > 0 ) return circleRectangleIntersection( r, 0, y1, x2,
           y2 ) + circleRectangleIntersection( r, x1, y1, 0, y2 );
14     if( y1 < 0 && y2 > 0 ) return circleRectangleIntersection( r, x1, 0, x2,
           y2 ) + circleRectangleIntersection( r, x1, y1, x2, 0 );
15     if( x1 < 0 ) return circleRectangleIntersection( r, -x2, y1, -x1, y2 );
16     if( y1 < 0 ) return circleRectangleIntersection( r, x1, -y2, x2, -y1 );
17     if( x1 >= r || y1 >= r ) return 0.0;
18     if( x2 > r ) return circleRectangleIntersection( r, x1, y1, r, y2 );
19     if( y2 > r ) return circleRectangleIntersection( r, x1, y1, x2, r );
20     if( x1*x1 + y1*y1 >= r*r ) return 0.0;
21     if( x2*x2 + y2*y2 <= r*r ) return (x2 - x1) * (y2 - y1);
22     int outCode = ( x2*x2 + y1*y1 >= r*r ) + 2 * ( x1*x1 + y2*y2 >= r*r );
23
24     if( outCode == 3 ) return areaArc( r, x1, y1 );
25     else if( outCode == 1 )
26     {
27         double x = sqrt( r*r - y2*y2 + 0.0 );
28         return (x - x1) * (y2 - y1) + areaArc( r, x, y1 );
29     }
30     else if( outCode == 2 )
31     {
32         double y = sqrt( r*r - x2*x2 + 0.0 );
33         return (x2 - x1) * (y - y1) + areaArc( r, x1, y );
34     }
35     else
36     {
37         double x = sqrt( r*r - y2*y2 + 0.0 );
38         double y = sqrt( r*r - x2*x2 + 0.0 );
39         return (x2 - x1) * (y - y1) + (x - x1) * (y2 - y) + areaArc( r, x, y )
               ;
40     }
41 }
42
43 const double pi = 2 * acos(0.0);
44
45 int cases, caseno;
46
47 struct circle
48 {
```

```
    int x, y, r;
};

double circleRectangleIntersection( circle C, int x1, int y1, int x2, int y2 )
{
    return circleRectangleIntersection( C.r, x1 - C.x, y1 - C.y, x2 - C.x, y2
        - C.y );
}
```

3.10 closest pair

```
// Closest Pair

bool cmp(const point &u, const point &v){
        if(eq(u.y , v.y)) return u.x < v.x;
        return u.y > v.y;
}

// Don't forget to sort all the points before calling closest_pair
// Don't forget to call unique() over the points
// Indexing does not matter here
long double closest_pair(point *P, int st, int ed)
{
    if(st == ed) return numeric_limits<double>::max();
    if(st + 1 == ed) return abs(P[st] -  P[ed]);

    int mid = (st+ed)/2, i, j, k, turn;
    long double soFar = min(closest_pair(P, st, mid), closest_pair(P, mid+1,
        ed));


    vector<point> Lt, Rt;
    for(i = st; i <= mid; i++)
        if(abs(P[mid].x - P[i].x) < soFar + eps)
            Lt.push_back(P[i]);

    for(i = mid+1; i <= ed; i++)
        if(abs(P[i].x - P[mid].x) < soFar + eps)
            Rt.push_back(P[i]);


    stable_sort(Lt.begin(), Lt.end(), cmp);
    stable_sort(Rt.begin(), Rt.end(), cmp);

    for(i = j = 0; i < Lt.size(); i++)
    {
        while(j < Rt.size() && Rt[j].y > Lt[i].y + soFar + eps) j++;
            for(k = j; k < min((int) Rt.size(), j + 6); k++) // You may
                increase the bound.
                soFar = min(soFar, (long double) abs(Lt[i] - Rt[k]));
    }

```

```
    return soFar;
}
```

3.11 complex things

```

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

#define pi      acos(-1.00)
#define eps     1e-9
#define D(x)    cout << #x " = " << (x) << endl

const int inf = numeric_limits<int>::max();
bool eq(double a, double b) { return fabs( a - b ) < eps; } //two numbers are equal

/*
Good to know:
-> Cross product is distributive over addition: a x (b + c) = a x b + a x c
-> Cross product is anticommutative  a x b = -(b x a)
-> Cross product is not associative a x ( b x c ) != (a x b) x c
-> Dot product is distributive over addition: A.(B+C) = A.B + A.C.
*/

struct point{
    double x, y;
    point(){}

    point(double xx, double yy) {x = xx, y = yy;} // NEVER USE xx = 0 or yy = 0 HERE
} origin = point(0, 0);

struct line { // Creates a line with equation ax + by + c = 0
    double a, b, c;
    point p1, p2;

    line() {}
    line(double _a, double _b, double _c){
        a = _a, b = _b, c = _c;
        assert( !(eq(a, 0) && eq(b, 0) ) );

        if(eq(a, 0)) p1 = point(0, -c/b), p2 = point(1, -c/b);
        else if(eq(b, 0)) p1 = point( -c/a, 0), p2 = point( -c/a, 1);
        else p1 = point(0, -c/b), p2 = point(-c/a, 0);
    }
    line( point _p1,point _p2 ) {
        p1 = _p1, p2 = _p2;

        a = p1.y - p2.y;
        b = p2.x - p1.x;
        c = p1.x * p2.y - p2.x * p1.y;
```

```
46      }
47  };
48
49  point operator+(const point &u, const point &v) {return point(u.x + v.x, u.y +
        v.y);} //OK
50  point operator-(const point &u, const point &v) {return point(u.x - v.x, u.y -
        v.y);} //OK
51  point operator*(const point &u, const point &v) {return point(u.x * v.x - u.y
        * v.y, u.x * v.y + v.x * u.y);} // multiplying two complex numbers
52  point operator*(const point &u, double v) {return point(u.x*v, u.y*v);} //OK
53  point operator*(double v, const point &u) {return point(u.x*v, u.y*v);} //OK
54  point operator/(const point &u, double v) {assert(abs(v) > eps); return point(
        u.x/v, u.y/v);} //OK
55  bool operator != (const point &u, const point &v) {return !(eq(u.x, v.x) && eq
        (u.y, v.y));} //OK
56
57  bool operator <(const point &u, const point &v){
58      if(fabs(u.x - v.x ) < eps) return u.y + eps < v.y;
59      return u.x + eps < v.x;
60  }
61
62  double norm(point u){return sqrt(u.x * u.x + u.y * u.y);} //OK
63  double arg(point u){ assert(u != origin); return atan2(u.y, u.x);} //OK
64  point polar(double r, double theta) {return point(r * cos(theta), r * sin(
        theta));} //OK
65
66  double dotp(point u, point v) {return u.x * v.x + u.y * v.y;} //OK
67  double crsp(point u, point v) {return u.x * v.y - u.y * v.x;} //OK
68
69  double smlr_angle(point l, point m, point r) {return abs(remainder(arg(l-m) -
        arg(r-m), 2.0 * pi)); } //OK
70
71  point unit_vector(point u) { return u / norm(u); } //OK
72  point rtt(point piv, point u, double theta) {return (u - piv) * polar(1.00,
        theta) + piv;} //OK
73  point projection(point p, point st, point ed) { return dotp(ed - st, p - st) /
         norm(ed - st) * unit_vector(ed - st) + st;} //OK
74  point extend(point st, point ed, double len) { return ed + unit_vector(ed-st)
        * len;} //OK
75
76  point segmentProjection(point p, point st, point ed)
77  {
78      double d = dotp(p - st, ed - st) / norm(ed - st);
79      if(d < 0) return st;
80      if(d > norm(ed - st) + eps) return ed;
81      return st + unit_vector(ed - st) * d;
82  } //OK
83
84  double distancePointSegment(point p, point st, point ed) {return norm(p -
        segmentProjection(p, st, ed)); } //OK
85  double distancePointLine( point P, point st, point ed) { return norm(
```

```
    projection(P, st, ed) - P ); } //OK
86
87  point reflection(point p, point st, point ed){
88      point proj = projection(p, st, ed);
89      if(p != proj) return extend(p, proj, norm(p - proj));
90      return proj;
91  } //OK
92
93  bool collinear(point p, point st, point ed) {return fabs(crsp(p - st, ed -st))
        < eps; }
94
95  int lineLineIntersection(point a, point b, point p, point q, point &ret){
96      if(fabs(crsp(b - a, p - q)) < eps){
97          if(collinear(a, p, q)) { ret = a; return inf; }
98          return 0;
99      }
100     else {
101         ret = a + (b - a) * crsp(p - a, p - q) / crsp(b - a, p - q);
102         return 1;
103     }
104 }
105
106 int lineLineIntersection(line L1, line L2, point &ret){
107     return lineLineIntersection(L1.p1, L1.p2, L2.p1, L2.p2, ret);
108 }
109
110 bool segmentSegmentIntersection(point a, point b, point p, point q, point &ret
    )
111 {
112     if( fabs(crsp(b - a, q - p)) < eps ) {
113         if( eq(norm(a - p) + norm(q - a), norm(q - p) ) ) {ret = a; return
              true;}
114         if( eq(norm(b - p) + norm(q - b), norm(q - p) ) ) {ret = b; return
              true;}
115         return false;
116     }
117
118     double dir1 = crsp(b - a, p - a), dir2 = crsp(b - a, q - a);
119     if( (dir1 + eps < 0 && dir2 + eps < 0) || (dir1 > eps && dir2 > eps) )
          return false;
120
121     dir1 = crsp(q - p, a - p), dir2 = crsp(q - p, b - p);
122     if( (dir1 + eps < 0 && dir2 + eps < 0) || (dir1 > eps && dir2 > eps) )
          return false;
123
124     return lineLineIntersection(a, b, p, q, ret);
125 }
126
127 point circumCircleCenter(point u, point v, point w)
128 {
129     assert(collinear(u, v, w) == false);
```

```
    point vw_mid = (v + w) / 2;
    point uv_mid = (u + v) / 2;

    point ret;
    lineLineIntersection(vw_mid, rtt(vw_mid, extend(v, vw_mid, 1), pi/2),
                         uv_mid, rtt(uv_mid, extend(u, uv_mid, 1), pi/2), ret)
                            ;

    return ret;
}

double angle(point u, point v) { return acos(dotp(u, v) / (norm(u) * norm(v)))
   ;}


///OK?
bool inDisk(point a, point b, point p) { return dotp(a-p, b-p) <= eps;}
bool onSegment(point x, point l, point r){
    return eq(crsp(l - x, r - x), 0) && inDisk(l, r, x);
}

bool segSegIntersection(point u, point v, point a, point b){
    double ang1 = crsp(a - v, u - v);
    double ang2 = crsp(b - v, u - v);
    if(ang1 > eps && ang2 > eps) return false;
    if(ang1 < -eps && ang2 < -eps) return false;

    if(fabs(ang1) < eps || fabs(ang2) < eps)
    {
        if(fabs(ang1) < eps){
            if(onSegment(a, u, v)) return true;
        }
        if(fabs(ang2) < eps){
            if(onSegment(b, u, v)) return true;
        }

        return false;
    }

    swap(u, a);
    swap(v, b);
    ang1 = crsp(a - v, u - v);
    ang2 = crsp(b - v, u - v);
    if(ang1 > eps && ang2 > eps) return false;
    if(ang1 < -eps && ang2 < -eps) return false;

    if(fabs(ang1) < eps || fabs(ang2) < eps)
    {
        if(fabs(ang1) < eps){
            if(onSegment(a, u, v)) return true;
        }
```

```
        if(fabs(ang2) < eps){
            if(onSegment(b, u, v)) return true;
        }

        return false;
    }

    return true;
}

struct vline{
    point v;
    double c;

    vline(double a, double b, double _c){ /// a * x + b * y + _c = 0
        /*
            Let (-b, a) be a vector
            (-b, a) x (x, y) = -(_c) when (x, y) belongs to the line

            INVARIENT: v x (any_point_on_the_line) = c
        */
        v = point(-b, a);
        c = -(_c);
    }
    vline(point _v, double _c){
        v = _v;
        c = _c;
    }
    vline(point p, point q){
        v = q - p;
        c = crsp(v, p);
    }

    line get_line(){ return line(v.y, -v.x, -c); }
};

vline translate(vline l, point t) {return vline(l.v, l.c + crsp(l.v, t));}

vline angBisector(vline l1, vline l2, bool interior) {
    assert(!(eq(crsp(l1.v, l2.v), 0)));
    double sign = interior ? 1 : -1;
    return {l2.v/norm(l2.v) + sign * l1.v/norm(l1.v), l2.c/norm(l2.v) + sign *
        l1.c/norm(l1.v)};
}


int main()
{
    point p = point(0, 10);
    point q = point(10, 0);

```

```
    vline l1 = vline(origin, p);
    vline l2 = vline(origin, q);
    vline l = angBisector(l1, l2, true);
    line x = l.get_line();
    D(x.a);
    D(x.b);
    D(x.c);

    point r = point(5, 5);
    D(onSegment(r, p, q));

    return 0;
}
```

3.12 euclidean things

```

//Euclidean things:
const double eps = 1e-11, pi = 2 * acos( 0.0 );

struct point { // Creates normal 2D point
    double x, y;
    point() {}
    point( double xx, double yy ) { x = xx, y = yy; }
};
struct point3D { // Creates normal 3D point
    double x, y, z;
};
struct line { // Creates a line with equation ax + by + c = 0
    double a, b, c;
    line() {}
    line( point p1,point p2 ) {
        a = p1.y - p2.y;
        b = p2.x - p1.x;
        c = p1.x * p2.y - p2.x * p1.y;
    }
};
struct circle { // Creates a circle with point 'center' as center and r as
    radius
    point center;
    double r;
    circle() {}
    circle( point P, double rr ) { center = P; r = rr; }
};
struct segment { // Creates a segment with two end points -> A, B
    point A, B;
    segment() {}
    segment( point P1, point P2 ) { A = P1, B = P2; }
};

inline bool eq(double a, double b) { return fabs( a - b ) < eps; } //two
    numbers are equal
```

```
35
36 //Distance - Point, Point:
37 inline double Distance( point a, point b ) {
38     return sqrt( ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y )
       );
39 }
40
41 //Distance^2 - Point, Point:
42 inline double sq_Distance( point a, point b ) {
43     return ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y );
44 }
45
46 //Distance - Point, Line:
47 inline double Distance( point P, line L ) {
48     return fabs( L.a * P.x + L.b * P.y + L.c ) / sqrt( L.a * L.a + L.b * L.b )
       ;
49 }
50
51 inline double isleft( point p0, point p1, point p2 ) {
52     return( ( p1.x - p0.x ) * ( p2.y - p0.y ) - ( p2.x - p0.x ) * ( p1.y - p0.
       y ) );
53 }
54
55 //Intersection - Line, Line:
56 inline bool intersection( line L1, line L2, point &p ) {
57     double det = L1.a * L2.b - L1.b * L2.a;
58     if( eq ( det, 0 ) ) return false;
59     p.x = ( L1.b * L2.c - L2.b * L1.c ) / det;
60     p.y = ( L1.c * L2.a - L2.c * L1.a ) / det;
61     return true;
62 }
63
64 //Intersection - Segment, Segment:
65 inline bool intersection( segment L1, segment L2, point &p ) {
66     if( !intersection( line( L1.A, L1.B ), line( L2.A, L2.B ), p) ) {
67         return false; // can lie on another, just check their equations, and
           check overlap
68     }
69     return(eq(Distance(L1.A,p)+Distance(L1.B,p),Distance(L1.A,L1.B)) &&
70         eq(Distance(L2.A,p)+Distance(L2.B,p),Distance(L2.A,L2.B)));
71 }
72
73 //Perpendicular Line of a Given Line Through a Point:
74 inline line findPerpendicularLine( line L, point P ) {
75     line res; //line perpendicular to L, and intersects with P
76     res.a = L.b, res.b = -L.a;
77     res.c = -res.a * P.x - res.b * P.y;
78     return res;
79 }
80
81 //Distance - Point, Segment:
```

```
inline double Distance( point P, segment S ) {
    line L1 = line(S.A,S.B), L2; point P1;
    L2 = findPerpendicularLine( L1, P );
    if( intersection( L1, L2, P1 ) )
        if( eq ( Distance( S.A, P1 ) + Distance( S.B, P1 ), Distance( S.A, S.B ) ) )
            return Distance(P,L1);
    return min ( Distance( S.A, P), Distance( S.B, P) );
}

//Area of a 2D Polygon:
double areaPolygon( point P[], int n ) {
    double area = 0;
    for( int i = 0, j = n - 1; i < n; j = i++ ) area += P[j].x * P[i].y - P[j].y * P[i].x;
    return fabs(area)/2;
}

//Point Inside Polygon:
bool insidePoly( point &p, point P[], int n ) {
    bool inside = false;
    for( int i = 0, j = n - 1; i < n; j = i++ )
        if( (( P[i].x < p.x ) ^ ( P[j].x < p.x )) &&
(P[i].y - P[j].y) * abs(p.x - P[j].x) < (p.y - P[j].y) * abs(P[i].x - P[j].x) )
            inside = !inside;
    return inside;
}

//Intersection - Circle, Line:
inline bool intersection(circle C,line L,point &p1,point &p2) {
    if( Distance( C.center, L ) > C.r + eps ) return false;
    double a, b, c, d, x = C.center.x, y = C.center.y;
    d = C.r*C.r - x*x - y*y;
    if( eq( L.a, 0) ) {
        p1.y = p2.y = -L.c / L.b;
        a = 1;
        b = 2 * x;
        c = p1.y * p1.y - 2 * p1.y * y - d;
        d = b * b - 4 * a * c;
        d = sqrt( fabs (d) );
        p1.x = ( b + d ) / ( 2 * a );
        p2.x = ( b - d ) / ( 2 * a );
    }
    else {
        a = L.a *L.a + L.b * L.b;
        b = 2 * ( L.a * L.a * y - L.b * L.c - L.a * L.b * x);
        c = L.c * L.c + 2 * L.a * L.c * x - L.a * L.a * d;
        d = b * b - 4 * a * c;
        d = sqrt( fabs(d) );
        p1.y = ( b + d ) / ( 2 * a );
```

```
130         p2.y = ( b - d ) / ( 2 * a );
131         p1.x = ( -L.b * p1.y -L.c ) / L.a;
132         p2.x = ( -L.b * p2.y -L.c ) / L.a;
133     }
134     return true;
135 }
136
137 //Find Points that are r1 unit away from A, and r2 unit away from B:
138 inline bool findpointAr1Br2(point A,double r1,point B, double r2,point &p1,
        point &p2) {
139     line L;
140     circle C;
141     L.a = 2 * (B.x - A.x );
142     L.b = 2 * (B.y - A.y );
143     L.c = A.x * A.x + A.y * A.y - B.x * B.x - B.y * B.y + r2 * r2 - r1 * r1;
144     C.center = A;
145     C.r = r1;
146     return intersection( C, L, p1, p2 );
147 }
148
149 //Intersection Area between Two Circles:
150 inline double intersectionArea2C( circle C1, circle C2 ) {
151     C2.center.x = Distance( C1.center, C2.center );
152     C1.center.x = C1.center.y = C2.center.y = 0;
153     if( C1.r < C2.center.x - C2.r + eps ) return 0;
154     if( -C1.r + eps > C2.center.x - C2.r ) return pi * C1.r * C1.r;
155     if( C1.r + eps > C2.center.x + C2.r ) return pi * C2.r * C2.r;
156     double c, CAD, CBD, res;
157     c = C2.center.x;
158     CAD = 2 * acos( (C1.r * C1.r + c * c - C2.r * C2.r) / (2 * C1.r * c) );
159     CBD = 2 * acos( (C2.r * C2.r + c * c - C1.r * C1.r) / (2 * C2.r * c) );
160     res=C1.r * C1.r * ( CAD - sin( CAD ) ) + C2.r * C2.r * ( CBD - sin ( CBD )
            );
161     return .5 * res;
162 }
163
164 //Circle Through Thee Points:
165 circle CircleThrough3points( point A, point B, point C) {
166     double den; circle c;
167     den = 2.0 *((B.x-A.x)*(C.y-A.y) - (B.y-A.y)*(C.x-A.x));
168     c.center.x = ( (C.y-A.y)*(B.x*B.x+B.y*B.y-A.x*A.x-A.y*A.y) -(B.y-A.y)*(C.x
            *C.x+C.y*C.y-A.x*A.x-A.y*A.y) );
169     c.center.x /= den;
170     c.center.y =( (B.x-A.x)*(C.x*C.x+C.y*C.y-A.x*A.x-A.y*A.y) - (C.x-A.x)*(B.x
            *B.x+B.y*B.y-A.x*A.x-A.y*A.y) );
171     c.center.y /= den;
172     c.r = Distance( c.center, A );
173     return c;
174 }
175
176 //Rotating a Point anticlockwise by 'theta' radian w.r.t Origin:
```

```
177 inline point rotate2D( double theta, point P ) {
178     point Q;
179     Q.x = P.x * cos( theta ) - P.y * sin( theta );
180     Q.y = P.x * sin( theta ) + P.y * cos( theta );
181     return Q;
182 }
183 
184 
185 // Checks whether ractangle with sides (a, b) fits into rectangle with sides (
        c, d)
186 bool fits( int a, int b, int c, int d ) {
187     double X, Y, L, K, DMax;
188     if( a < b ) swap( a, b );
189     if( c < d ) swap( c, d );
190     if( c <= a && d <= b ) return true;
191     if( d >= b ) return false;
192     X = sqrt( a*a + b*b );
193     Y = sqrt( c*c + d*d );
194     if( Y < b ) return true;
195     if( Y > X ) return false;
196     L = ( b - sqrt( Y*Y - a*a) ) /2;
197     K = ( a - sqrt( Y*Y - b*b) ) /2;
198     DMax = sqrt(L * L + K * K);
199     if( d >= DMax ) return false;
200 return true;
201 }
202 
203 //Covex Hull
204 // compare Function for qsort in convex hull
205 point Firstpoint;
206 int cmp(const void *a,const void *b) {
207     double xx,yy;
208     point aa,bb;
209     aa = *(point *)a;
210     bb = *(point *)b;
211     xx = isleft( Firstpoint, aa, bb );
212     if( xx > eps ) return -1;
213     else if( xx < -eps ) return 1;
214     xx = sq_Distance( Firstpoint, aa );
215     yy = sq_Distance( Firstpoint, bb );
216     if( xx + eps < yy ) return -1;
217     return 1;
218 }
219 // 'P' contains all the points, 'C' contains the convex hull
220 // 'nP' = total points of 'P', 'nC' = total points of 'C'
221 void ConvexHull( point P[], point C[], int &nP, int &nC ) {
222     int i, j, pos = 0; // Remove duplicate points if necesary
223     for( i = 1; i < nP; i++ )
224         if( P[i].y < P[pos].y || ( eq( P[i].y, P[pos].y ) && P[i].x > P[pos].x
              + eps ) )
225             pos = i;
```

```
    swap( P[pos], P[0] );
    Firstpoint = P[0];
    qsort( P + 1, nP - 1, sizeof( point ), cmp );
    C[0] = P[0]; C[1] = P[1];
    i = 2, j = 1;
    while( i < nP ) {
        if( isleft( C[j-1], C[j], P[i] ) > -eps ) C[++j] = P[i++];
        else j--;
    }
    nC = j + 1;
}
```

3.13 inConvexPoly

```

/*
InConvexPoly:
P contains points in acw order. Works for only convex polygon
Complexity O(lg n)
*/

LL triArea2(pii a, pii b, pii c) // includes sign
{
    LL ret = 0;
    ret += (LL) a.xx*b.yy + (LL) b.xx*c.yy + (LL) c.xx*a.yy - (LL) a.xx*c.yy -
         (LL) c.xx*b.yy - (LL) b.xx*a.yy;
    return ret;
}

bool inConvexPoly(vector<pair <int, int> > &P, pair <int, int> q)
{
    pii fix = P[0];
    int st = 1, ed = P.size()-1, mid;

    while(ed - st > 1)
    {
        mid = (st+ed)>>1;
        if(triArea2(fix, P[mid], q) > 0 ) st = mid;
        else ed = mid;
    }

    if(triArea2(fix, P[st], q) < 0) return false;
    if(triArea2(P[st], P[ed], q) < 0) return false;
    if(triArea2(P[ed], fix, q) < 0) return false;
    return true;
}
```

3.14 maxPointCoverWithACricleOfRadiusR

```

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

```
4  #define sf(n)        scanf("%d", &n)
5  #define sff(a,b)     scanf("%d %d", &a, &b)
6  #define sfff(a,b,c)  scanf("%d %d %d", &a, &b, &c)
7  #define pb           push_back
8  #define mp           make_pair
9  #define xx           first
10 #define yy           second
11 #define MAX          2000
12 #define eps          1e-9
13 #define pi           acos(-1.00)
14
15 struct point{
16     double x, y;
17     point(){}
18     point(double xx, double yy){x = xx, y = yy;}
19 };
20
21 point operator+(const point &u, const point &v){return point(u.x + v.x, u.y +
       v.y);}
22 point operator-(const point &u, const point &v){return point(u.x - v.x, u.y -
       v.y);}
23 bool operator == (const point &u, const point &v) {return fabs(u.x - v.x) <
       eps && fabs(u.y - v.y ) < eps;}
24
25 double norm(point u) {return u.x*u.x + u.y*u.y;}
26 double abs(point u) {return sqrt(norm(u));}
27 double arg(point u) {return atan2(u.y, u.x);}
28 double get_angle(double a, double b, double c) {return acos( (b*b+c*c-a*a)/(2*
       b*c) ); }
29
30 bool cmp(pair< double, int> u, pair< double, int> v)
31 {
32     if(fabs(u.xx - v.xx) < eps) return u.yy > v.yy + eps;
33     return u.xx + eps < v.xx;
34 }
35
36 int getMaxInt(vector < pair< double, int> > &vec)
37 {
38     stable_sort(vec.begin(), vec.end(), cmp);
39     int ret = 0, mx = 0;
40     for(auto x : vec)
41     {
42         mx += x.yy;
43         ret = max(ret, mx);
44     }
45
46     return ret;
47 }
48
49 int maxPointCover(double radius, point pnt[], int n)
50 {
```

```
    int i, j, ret = (bool) n, cnt;
    vector< pair< double, int> > ep[2];

    for(i = 1; i <= n; i++)
    {
        ep[0].clear();
        ep[1].clear();
        cnt = 0;

        for(j = 1; j <= n; j++)
        {
            if(pnt[i] == pnt[j]){
                cnt++;
                continue;
            }
            if(abs(pnt[j] - pnt[i]) > 2 * radius + eps) continue;

            double ang = get_angle(radius, abs(pnt[j] - pnt[i]), radius);
            double curr = arg(pnt[j] - pnt[i]);

            double seg_st = remainder(curr - ang, 2*pi);
            double seg_ed = remainder(curr + ang, 2*pi);

            if(seg_st + eps < 0 && seg_ed > eps){
                ep[0].pb(mp(0, +1));
                ep[0].pb(mp(abs(seg_st), -1));

                ep[1].pb(mp(0, +1));
                ep[1].pb(mp(seg_ed, -1));
            }
            else if(seg_st > eps && seg_ed + eps < 0){
                ep[0].pb(mp(abs(seg_ed), +1));
                ep[0].pb(mp(pi, -1));

                ep[1].pb(mp(seg_st, +1));
                ep[1].pb(mp(pi, -1));
            }

            else if(seg_st > eps)
            {
                ep[1].pb(mp(seg_st, +1));
                ep[1].pb(mp(seg_ed, -1));
            }
            else
            {
                ep[0].pb(mp(abs(seg_st), +1));
                ep[0].pb(mp(abs(seg_ed), -1));
            }
        }

        cnt += max(getMaxInt(ep[0]), getMaxInt(ep[1]));
```

```
        ret = max(ret, cnt);
    }

    return ret;
}

point P[MAX+5];

int main()
{
    //freopen("in.txt", "r", stdin);
    int i, j, k, n,r;
    int x, y;

    while(sff(n, r) == 2 && (n || r))
    {
        for(i = 1; i <= n; i++)
        {
            sff(x, y);
            P[i] = point(x, y);
        }
        printf("It is possible to cover %d points.\n", maxPointCover(r, P, n))
            ;
    }
    return 0;
}
```

3.15 minimum enclosing circle

```

//Minimum Enclosing Sphere

struct point3D{
    double x, y, z;
    point3D(){}
    point3D(double xx, double yy, double zz):x(xx),y(yy),z(zz){}
}P[MAX+5];

struct sphere{
    point3D center;
    double r;
    sphere(){}
    sphere(point3D p, double rr):center(p), r(rr){}
};

double abs(double x, double y, double z)
{
    return (x*x+y*y+z*z);
}

sphere minimumEnclosingSphere(point3D arr[], int n) // 1 based indexing
{
```

```
    point3D piv = point3D(0,0,0);

    int i, j;
    for(i = 1; i <= n; i++)
    {
        piv.x += arr[i].x;
        piv.y += arr[i].y;
        piv.z += arr[i].z;
    }

    piv.x /= n;
    piv.y /= n;
    piv.z /= n;

    double p = 0.1, e, d;

    for (i = 0; i < 70000; i++) // better to have 50K+
    {
        int f = 0;
        d = numeric_limits<double>::min();
        for (j = 1; j <= n; j++)
        {
            e = abs(piv.x - arr[j].x, piv.y - arr[j].y, piv.z - arr[j].z);
            if (d < e) {
                d = e;
                f = j;
            }
        }

        piv.x += (arr[f].x - piv.x)*p;
        piv.y += (arr[f].y - piv.y)*p;
        piv.z += (arr[f].z - piv.z)*p;
        p *= 0.998;
    }

    return sphere(piv, sqrt(d));
}
```

3.16 miscellaneous

```

//3_point_orientation
LL triArea2(pii a, pii b, pii c) // includes sign
{
    LL ret = 0;
    ret += (LL) a.xx* (b.yy - c.yy) + (LL) b.xx*(c.yy - a.yy) + (LL) c.xx*(a.
       yy - b.yy);
    return ret;
}


//Angle between 3 points + Dot product
```

```
double ang(pii L, pii mid, pii R)
{
    double dot = (LL) (L.xx - mid.xx) * (R.xx - mid.xx) + (LL) (L.yy - mid.yy)
         * (R.yy - mid.yy);
    dot /= sqrt((L.xx - mid.xx) * (L.xx - mid.xx) + (L.yy - mid.yy) * (L.yy -
       mid.yy));
    dot /= sqrt((R.xx - mid.xx) * (R.xx - mid.xx) + (R.yy - mid.yy) * (R.yy -
       mid.yy));
    return acos(dot);
}


//height of trapezium_from_sides
double tri_area(double a,double b,double c)
{
    double s=(a+b+c)/2;
    return sqrt(s*(s-a)*(s-b)*(s-c));
}
double height_of_trapezium_from_sides(double a,double b, double c, double d)
{
    double h;
    if(a<c) swap(a,c);
    h=2*tri_area(a,b,d)/(a-c);
    return h;
}
```

3.17 seg seg intersection

```

///Segment Segment Intersection (2D)
int order(pii st, pii ed, pii q){
    LL xp = (LL) (ed.xx - st.xx) * (q.yy - ed.yy) - (LL) (ed.yy - st.yy) * (q.
       xx - ed.xx);
    if(!xp) return 0;
    if(xp > 0) return 1;
    return -1;
}

bool onsegment(pii l, pii r, pii mid) // (l___r) -> mid
{
    if(order(l,r,mid)) return 0;
    return (min(l.xx,r.xx) <= mid.xx && mid.xx <= max(l.xx,r.xx) && min(l.yy,r
       .yy) <= mid.yy && mid.yy <= max(l.yy,r.yy));
}

inline bool intersect(pii p1, pii p2, pii p3, pii p4)
{
    int d1, d2, d3, d4;
    d1 = order(p3, p4, p1);
    d2 = order(p3, p4, p2);
    d3 = order(p1, p2, p3);
    d4 = order(p1, p2, p4);
```

```
    if(((d1 < 0 && d2 > 0) || (d1 > 0 && d2 < 0)) && ((d3 < 0 && d4 > 0) || (
        d3 > 0 && d4 < 0))) return true;
    if(!d3 && onsegment(p1, p2, p3)) return true;
    if(!d4 && onsegment(p1, p2, p4)) return true;
    if(!d1 && onsegment(p3, p4, p1)) return true;
    if(!d2 && onsegment(p3, p4, p2)) return true;
    return false;
}
```

3.18 tangent of two circles

```

//Tangent of two Circles
const double eps = 1e-10;
const double pi = acos(-1);

int dcmp(double x){
    return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1);
}

struct Point{
    double x;
    double y;
    Point(double x = 0, double y = 0):x(x), y(y){}

    bool operator < (const Point& e) const{
        return dcmp(x - e.x) < 0 || (dcmp(x - e.x) == 0 && dcmp(y - e.y) < 0);
    }
    int read(){
        return scanf("%lf%lf", &x, &y);
    }
}p[3];

typedef Point Vector;

Vector operator + (Point A, Point B)    { return Vector(A.x + B.x, A.y + B.y)
    ;}
Vector operator - (Point A, Point B)    { return Vector(A.x - B.x, A.y - B.y)
    ;}
Vector operator * (Point A, double p)   { return Vector(A.x * p, A.y * p);}
Vector operator / (Point A, double p)   { return Vector(A.x / p, A.y / p);}
double Dot(Vector A, Vector B)          { return A.x * B.x + A.y * B.y;}
double Cross(Vector A, Vector B)        { return A.x * B.y - B.x * A.y;}
double Length(Vector A)                 { return sqrt(Dot(A, A));}

struct Circle{
    double x, y;
    double r;
    Circle(double x = 0, double y = 0, double r = 0):x(x), y(y), r(r){}

    int read(){
        return scanf("%lf%lf%lf", &x, &y, &r);
```

```
    }
    Point point(double a){
        return Point(x + r * cos(a), y + r * sin(a));
    }
    double getmax(){return max(max(x,y),r); }
};
//returns -1 when infinite number of tangents
int getTangents(Circle A, Circle B, Point *a, Point *b)
{
    int cnt = 0;
    if(dcmp(A.r - B.r) < 0){
        swap(A, B);
        swap(a, b);
    }
    double d = sqrt((A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y));
    double rdiff = A.r - B.r;
    double rsum = A.r + B.r;
    if(dcmp(d - rdiff) < 0) return 0;
    double base = atan2(B.y - A.y, B.x - A.x);
    if(dcmp(d) == 0) return -1;
    if(dcmp(d - rdiff) == 0)
    {
        a[cnt] = b[cnt] = A.point(base);
        cnt++;
        return 1;
    }
    double ang = acos((A.r - B.r) / d);
    a[cnt] = A.point(base + ang); b[cnt] = B.point(base + ang); cnt++;
    a[cnt] = A.point(base - ang); b[cnt] = B.point(base - ang); cnt++;
    if(dcmp(d - rsum) == 0)
    {
        a[cnt] = b[cnt] = A.point(base);
        cnt++;
    }
    else if(dcmp(d - rsum) > 0)
    {
        double ang = acos((A.r + B.r) / d);
        a[cnt] = A.point(base + ang); b[cnt] = B.point(pi + base + ang); cnt
            ++;
        a[cnt] = A.point(base - ang); b[cnt] = B.point(pi + base - ang); cnt
            ++;
    }
    return cnt;
}
```

4 Graph Theory

4.1 2 sat samiul vai

```
#define SCCNODE adf
struct SCC{
    int num[SCCNODE], low[SCCNODE], col[SCCNODE], cycle[SCCNODE], st[SCCNODE];
```

```
4       int tail, cnt, cc;
5       vi adj[SCCNODE];
6
7       SCC():tail(0),cnt(0),cc(0) {}
8       void clear ( int n ) {
9           cc += 3;
10          FOR(i,0,n) adj[i].clear();
11          tail = 0;
12      }
13      void tarjan ( int s ) {
14          num[s] = low[s] = cnt++;
15          col[s] = cc + 1;
16          st[tail++] = s;
17          FOR(i,0,SZ(adj[s])-1) {
18              int t = adj[s][i];
19              if ( col[t] <= cc ) {
20                  tarjan ( t );
21                  low[s]=min(low[s],low[t]);
22              }
23              /*Back edge*/
24              else if (col[t]==cc+1)
25                  low[s]=min(low[s],low[t]);
26          }
27          if ( low[s] == num[s] ) {
28              while ( 1 ) {
29                  int temp=st[tail-1];
30                  tail--;
31                  col[temp] = cc + 2;
32                  cycle[temp] = s;
33                  if ( s == temp ) break;
34              }
35          }
36      }
37      void shrink( int n ) {
38          FOR(i,0,n){
39              FOR(j,0,SZ(adj[i])-1){
40                  adj[i][j] = cycle[adj[i][j]]; ///Careful. This will create
                        self-loop
41              }
42          }
43          FOR(i,0,n){
44              if ( cycle[i] == i ) continue;
45              int u = cycle[i];
46              FOR(j,0,SZ(adj[i])-1){
47                  int v = adj[i][j];
48                  adj[u].pb ( v );
49              }
50              adj[i].clear();
51          }
52          FOR(i,0,n){ ///Not always necessary
53              sort ( ALL(adj[i]) );
```

```
            UNIQUE(adj[i]);
        }
    }
    void findSCC( int n ) {
        FOR(i,0,n) {
            if ( col[i] <= cc ) {
                tarjan ( i );
            }
        }
    }
};

/*
1. The nodes need to be split. So change convert() accordingly.
2. Using clauses, populate scc edges.
3. Call possible, to find if a valid solution is possible or not.
4. Dont forget to keep space for !A variables
*/
struct SAT2 {
    SCC scc;

    SAT2(): bfscc(1) {}

    void clear( int n ) {
        scc.clear( int n );
    }

    int convert ( int n ) { ///Change here. Depends on how input is provided
        int x = ABS(n);
        x--;
        x *= 2;
        if ( n < 0 ) x ^= 1;
        return x;
    }

    void mustTrue ( int a ) { ///A is True
        scc.adj[a^1].pb ( a );
    }
    void orClause ( int a, int b ) { /// A || B clause
        //!a->b !b->a
        scc.adj[a^1].pb ( b );
        scc.adj[b^1].pb ( a );
    }
    /// Out of all possible option, only one is true
    void atMostOneClause ( int a[], int n, int flag ) {
        if ( flag == 0 ) { /// At most one can be false
            FOR(i,0,n){
                a[i] = a[i] ^ 1;
            }
        }
        FOR(i,0,n) {
```

```
            FOR(j,i+1,n) {
                orClause( a[i] ^ 1, a[j] ^ 1 ); /// !a || !b both being true not allowed
            }
        }
    }

    ///Send n, total number of nodes, after expansion
    bool possible( int n ) {
        scc.findSCC( n );

        FOR(i,0,n) {
            int a = i, b = i^1;
            ///Falls on same cycle a and !a.
            if ( scc.cycle[a] == scc.cycle[b] ) return false;
        }

        ///Valid solution exists
        return true;
    }

    ///To determine if A can be true. It cannot be true, if a path exists from A to !A.
    int vis[SAT2NODE], qqq[SAT2NODE], bfscc;
    void bfs( int s ) {
        bfscc++;
        int qs = 0, qt = 0;
        vis[s] = bfscc;
        qqq[qt++] = s;
        while ( qs < qt ) {
            s = qqq[qs++];
            FOR(i,0,SZ(scc.adj[s])-1) {
                int t = scc.adj[s][i];
                if ( vis[t] != bfscc ) {
                    vis[t] = bfscc;
                    qqq[qt++] = t;
                }
            }
        }
    }

}sat2;
```

4.2 MCMF with SPFA

```
// Works only on directed Graph
// *** 1 based indexing

#define MAXN    205
#define MAXE    100000
const int INF = 0x7f7f7f7f;

```

```
int src, snk, nNode, nEdge;
int fin[MAXN + 5], pre[MAXN + 5], dist[MAXN + 5];
int cap[2*MAXE+5], cost[2*MAXE+5], Next[2*MAXE+5], to[2*MAXE+5], from[2*MAXE
    +5];
bool inqueue[MAXN+5];

inline void init(int _src, int _snk, int nodes) {
    memset(fin, -1, sizeof(fin));
    nNode = nodes, nEdge = 0;
    src = _src, snk = _snk;
}

inline void addEdge(int u, int v, int _cost, int _cap) {
    from[nEdge] = u, to[nEdge] = v, cap[nEdge] = _cap, cost[nEdge] = _cost;
    Next[nEdge] = fin[u], fin[u] = nEdge++;
    from[nEdge] = v, to[nEdge] = u, cap[nEdge] = 0, cost[nEdge] = -(_cost);
    Next[nEdge] = fin[v], fin[v] = nEdge++;
    assert(nEdge <= 2*MAXE);
}

bool bellman() {
    int u, v, i;
    memset(dist, 0x7f, sizeof(dist));
    memset(pre, -1, sizeof(pre));
    memset(inqueue, false, sizeof(inqueue));

    dist[src] = 0;
    queue<int> q;
    q.push(src);
    inqueue[src] = true;

    while(!q.empty()){
        u = q.front(); q.pop();
        inqueue[u] = false;

        for(i = fin[u]; i >= 0; i = Next[i]) {
            v = to[i];
            if(cap[i] && dist[v] > dist[u] + cost[i]) {
                dist[v] = dist[u] + cost[i];
                pre[v] = i;
                if(inqueue[v] == false){
                    q.push(v);
                    inqueue[v] = true;
                }
            }
        }
    }
    return (dist[snk] < INF);
}

int mcmf(LL &fcost) {
```

```
    int netflow, bot, u;
    netflow = fcost = 0;
    while(bellman()) {
        bot = INF;
        for(u = pre[snk]; u >= 0; u = pre[from[u]]) bot = min(bot, cap[u]);
        for(u = pre[snk]; u >= 0; u = pre[from[u]]) {
            cap[u] -= bot;
            cap[u^1] += bot;
            fcost += (LL) bot * cost[u];
        }
        netflow += bot;
    }
    return netflow;
}
```

4.3 PRUFER CODE

```
/*
  Tree to Prufer Code
  Works for both 0 and 1 indexed node numbering
  Complexity: O(VlogV)
 */

vector<int> treeToPruferCode (int nodes, vector<pair<int,int>> &edges) {
    unordered_set<int> neighbors[nodes+1]; // For each node, who is it's
        neighbor?

    for( int i = 0; i < edges.size(); i++ ) {
        pair<int,int> edge = edges[i];
        int u = edges[i].first; int v = edges[i].second;
        neighbors[u].insert(v);
        neighbors[v].insert(u);
    }

    priority_queue<int> leaves;
    for ( int i = 0; i <= nodes; i++ ) {
        if (neighbors[i].size() == 1 ) {
            leaves.push(-i); // Negating since we need min heap
        }
    }
    vector<int> pruferCode;
    int need = nodes - 2;
    while(need--) {
        int leaf = -leaves.top(); leaves.pop();
        int neighborOfLeaf = *(neighbors[leaf].begin());
        pruferCode.push_back(neighborOfLeaf);
        // Remove the leaf
        neighbors[neighborOfLeaf].erase(leaf);
        // The neighbor can become a new leaf
        if(neighbors[neighborOfLeaf].size() == 1) {
            leaves.push(-neighborOfLeaf);
        }
```

```
    }
    return pruferCode;
}

/*
  Prufer Code to Tree
  Complexity: O(VlogV)
 */

vector<pair<int,int>> pruferCodeToTree(vector<int> &pruferCode) {
    // Stores number count of nodes in the prufer code
    unordered_map<int,int> nodeCount;

    // Set of integers absent in prufer code. They are the leaves
    set<int> leaves;

    int len = pruferCode.size();
    int node = len + 2;

    // Count frequency of nodes
    for ( int i = 0; i < len; i++ ) {
        int t = pruferCode[i];
        nodeCount[t]++;
    }

    // Find the absent nodes
    for ( int i = 1; i <= node; i++ ) {
        if ( nodeCount.find ( i ) == nodeCount.end() ) leaves.insert ( i );
    }

    vector<pair<int,int>> edges;
    /*Connect Edges*/
    for ( int i = 0; i < len; i++ ){
        int a = prufer[i]; // First node

        //Find the smallest number which is not present in prufer code now
        int b = *leaves.begin(); // the leaf

        edges.push_back({a,b}); // Edge of the tree

        leaves.erase ( b ); // Remove from absent list
        nodeCount[a]--; // Remove from prufer code
        if ( nodeCount[a] == 0 ) leaves.insert ( a ); // If a becomes absent
    }

    // The final edge
    edges.push_back({*leaves.begin(), *leaves.rbegin()});
    return edges;
}
```

4.4 articulation point

```

//Articulation Point
VI edge[MAX+10];
int AP[MAX+10], d[MAX+10], vis[MAX+10],tm, low[MAX+10];
// tm = 0, memset(AP,0), memset(vis,0)
// call for every not visited not with dfs(index,-1)

void dfs(int idx,int par)
{
    int i, cur, child = 0;
    vis[idx] = true;
    low[idx] = d[idx] = ++tm;

    for(i = 0; i < (int) edge[idx].size(); i++)
    {
        cur = edge[idx][i];
        if(cur == par) continue;

        if(vis[cur]) low[idx] = min(low[idx], d[cur]);
        else
        {
            child++;
            dfs(cur,idx);
            low[idx] = min(low[idx], low[cur]);
            if(par != -1 && low[cur] >= d[idx]) AP[idx]++;
        }
    }
    if(par == -1 && child > 1) AP[idx] = child;
}
```

4.5 bipartite matching

```

// bipartite matching using dfs [ 1 indexing ]

int Left[MAX+10], Right[MAX+10];
bool vis[MAX+10];
vector<int> edge[MAX+10]; // Left side Graph

bool dfs(int idx)
{
    if(vis[idx]) return false;
    vis[idx] = 1;

    int i, nw, len = edge[idx].size();
    for(i = 0; i < len; i++)
    {
        nw = edge[idx][i];
        if(Right[nw] == -1)
        {
            Right[nw] = idx;
            Left[idx] = nw;
```

```
            return true;
        }
    }

    for(i = 0; i < len; i++)
    {
        nw = edge[idx][i];
        if(dfs(Right[nw]))
        {
            Left[idx] = nw;
            Right[nw] = idx;
            return true;
        }
    }

    return false;
}

int match(int can) // can = cardinality of left half
{
    int i, ret = 0;
    bool done;

    memset(Left, -1, sizeof(Left));
    memset(Right, -1, sizeof(Right));
    do{
        done = true;
        memset(vis, false, sizeof(vis));
        for(i = 1; i <= can; i++)
            if(Left[i] == -1 && dfs(i))
                done = false;
    }while(!done);

    for(i = 1; i <= can; i++) ret += (Left[i] != -1);
    return ret;
}
```

4.6 bridges

```

//Bridges
VI edge[MAX+10];
int low[MAX+10], d[MAX+10], tm;
bool vis[MAX+10];
vector<pair <int, int> > bridges;

void dfs(int idx, int par)
{
    vis[idx] = true;
    int i, cur;
    low[idx] = d[idx] = ++tm;

```

```
    for(i = 0; i < (int) edge[idx].sz ; i++)
    {
        cur = edge[idx][i];
        if(cur == par) continue;

        if(vis[cur]) low[idx] = min(low[idx], d[cur]);
        else
        {
            dfs(cur, idx);
            low[idx] = min(low[idx], low[cur]);
            if(low[cur] > d[idx]) bridges.push_back(make_pair(min(cur,idx),
                max(cur,idx)));
        }
    }
}
```

4.7 centroid decomposition

```

#define MAXLG   18
#define MAXN    100000
#define BLACK   0
#define WHITE   1
#define dt      first
#define indx    second
typedef pair<int, int> pii;

///1 Based Indexing
struct centroid_tree{
    int tab[MAXLG+3][MAXN+3], par[MAXN+3], depth[MAXN+3], subsize[MAXN+3], clr
        [MAXN+3];
    vector<int> child[MAXN+3], edge[MAXN+3], cost[MAXN+3];
    bool vis[MAXN+3];
    set < pii > available[MAXN+3];


    void dfs(int idx, int &r, int p = -1){
        subsize[idx] = 1;
        for(auto x : edge[idx]){
            if(x == p) continue;
            if(vis[x]){
                if(r == -1) r = x;
                else if(depth[x] > depth[r]) r = x;
                continue;
            }
            dfs(x, r, idx);
            subsize[idx] += subsize[x];
        }
    }

    void find(int idx, int &c, int n, int p = -1){
```

```
        int mx = 0;
        for(auto x : edge[idx]){
            if(x == p){
                mx = max(mx, n - subsize[idx]);
                continue;
            }
            else if(vis[x]) continue;
            mx = max(mx, subsize[x]);
            find(x, c, n, idx);
        }

        if(mx <= n/2) c = idx;
    }

    void preprocess(int idx, int r, int d = 0, int p = -1){
        tab[depth[r]][idx] = d;
        int i, v, w;
        for(i = 0; i < (int) edge[idx].size(); i++){
            v = edge[idx][i];
            w = cost[idx][i];

            if(v == p) continue;
            if(vis[v]) continue;
            preprocess(v, r, d + w, idx);
        }
    }

    void build(int n){
        int iter = 1;

        while(iter <= n){
            if(vis[iter]) iter++;
            else{
                int r = -1, c = -1;
                dfs(iter, r);
                find(iter, c, subsize[iter]);

                assert(c != -1);
                if(r != -1) {
                    child[r].push_back(c);
                    depth[c] = depth[r] + 1;
                }
                else depth[c] = 0;

                par[c] = r;
                vis[c] = true;
                preprocess(c, c);
            }
        }
    }

```

```
85
86      void toggle(int idx){
87          if(clr[idx] == BLACK){
88              int c = idx;
89              do{
90                  available[c].insert(pii(tab[depth[c]][idx], idx));
91                  c = par[c];
92              }while(c != -1);
93          }
94          else{
95              int c = idx;
96              do{
97                  available[c].erase(pii(tab[depth[c]][idx], idx));
98                  c = par[c];
99              }while(c != -1);
100         }
101         clr[idx] = 1 - clr[idx];
102     }
103
104     int query(int idx){
105         if(clr[idx] == WHITE) return 0;
106         int ret = numeric_limits<int>::max();
107
108         int c = idx;
109         do{
110             if(available[c].size())  ret = min(ret, (*available[c].begin()).dt
                    + tab[depth[c]][idx]);
111             c = par[c];
112         }while(c != -1);
113
114         if(ret == numeric_limits<int>::max()) return -1;
115         return ret;
116     }
117
118 } ctree;
```

4.8 dijkstra

```
1
2  //Dijkstra
3
4  vector<int> edge[MAX+10], cost[MAX+10];
5  int dist[MAX+10];
6
7  struct node{
8      int idx, dt;
9      node(){};
10     node(int i, int d) {idx = i, dt = d;}
11 };
12
13 bool operator < (const node &a, const node &b) {return (a.dt > b.dt);}
14 priority_queue <node> Q;
```

```

void dijkstra(node src)
{
    memset(dist, -1, sizeof(dist));

    node nw;
    int i, u, v, c;

    dist[src.idx] = src.dt = 0;
    Q.push(src);
    while(!Q.empty())
    {
        nw = Q.top(); Q.pop();
        u = nw.idx;

        for(i = 0; i< (int) edge[u].size(); i++)
        {
            v = edge[u][i];
            c = cost[u][i];
            if(dist[v] == -1 || dist[v] > dist[u] + c) dist[v] = dist[u] + c,
               Q.push(node(v, dist[u]+c));
        }
    }

    return;
}
```

4.9 dinitz

```

/*
max flow (dinitz algorithm)
works on undirected/directed graph
**************in order to make it work for directed graph, change the add
   function*************
can have loops, multiple edges, cycles
DO NOT USE 0 as source, use 1 :)
INF has to be greater or equal to the max capacity in the network
*/

#define MAXN        200
#define MAXE        5000
const int INF = 0x7f7f7f7f;

int src, snk, nNode, nEdge;
int Q[MAXN+5], fin[MAXN+5], pro[MAXN+5], dist[MAXN+5];
int flow[2*MAXE+5], cap[2*MAXE+5], nxt[2*MAXE+5], to[2*MAXE+5];

void init(int _src, int _snk, int _n) {
    src = _src, snk = _snk, nNode = _n, nEdge = 0;
    memset(fin,-1,sizeof(fin));
}
```

```
23
24  void add_edge(int u, int v, int c) {
25      to[nEdge] = v, cap[nEdge] = c, flow[nEdge] = 0, nxt[nEdge] = fin[u], fin[u
            ] = nEdge++;
26      to[nEdge] = u, cap[nEdge] = c, flow[nEdge] = 0, nxt[nEdge] = fin[v], fin[v
            ] = nEdge++;
27      assert(nEdge <= 2*MAXE);
28  }
29
30  bool bfs() {
31      int st, en, i, u, v;
32      memset(dist, -1, sizeof(dist));
33      dist[src] = st = en = 0;
34      Q[en++] = src;
35      while(st < en) {
36          u = Q[st++];
37          for(i=fin[u]; i>=0; i=nxt[i]) {
38              v = to[i];
39              if(flow[i] < cap[i] && dist[v]==-1) {
40                  dist[v] = dist[u]+1;
41                  Q[en++] = v;
42              }
43          }
44      }
45      return dist[snk]!=-1;
46  }
47
48  int dfs(int u, int fl) {
49      if(u==snk) return fl;
50      for(int &e=pro[u], v, df; e>=0; e=nxt[e]) {
51          v = to[e];
52          if(flow[e] < cap[e] && dist[v]==dist[u]+1) {
53              df = dfs(v, min(cap[e]-flow[e], fl));
54              if(df>0) {
55                  flow[e] += df;
56                  flow[e^1] -= df;
57                  return df;
58              }
59          }
60      }
61      return 0;
62  }
63
64  LL dinitz() {
65      LL ret = 0;
66      int df;
67      while(bfs()) {
68          for(int i=1; i<=nNode; i++) pro[i] = fin[i];
69          while(true) {
70              df = dfs(src, INF);
71              if(df) ret += (LL)df;
```

```
            else break;
        }
    }
    return ret;
}
```

4.10 dinitz for double valued capacity

```

/*
max flow (dinitz algorithm)
works on undirected/directed graph
in order to make it work for directed graph, change the add function
can have loops, multiple edges, cycles
DO NOT USE 0 as source, use 1 :)
INF has to be greater than or equal to the max capacity in the network.
*/

#define MAXN        10110
#define MAXE        30110
const long double INF = 1e12;

int src, snk, nNode, nEdge;
int Q[MAXN+5], fin[MAXN+5], pro[MAXN+5], dist[MAXN+5];
long double flow[2*MAXE+5], cap[2*MAXE+5];
int nxt[2*MAXE+5], to[2*MAXE+5];

 void init(int _src, int _snk, int _n) {
    src = _src, snk = _snk, nNode = _n, nEdge = 0;
    memset(fin,-1,sizeof(fin));
}

 void add_edge(int u, int v, long double c) {
    to[nEdge] = v, cap[nEdge] = c, flow[nEdge] = 0, nxt[nEdge] = fin[u], fin[u
        ] = nEdge++;
    to[nEdge] = u, cap[nEdge] = 0.00, flow[nEdge] = 0, nxt[nEdge] = fin[v],
        fin[v] = nEdge++;
    assert(nEdge <= 2*MAXE);
}

bool bfs() {
    int st, en, i, u, v;
    memset(dist, -1, sizeof(dist));
    dist[src] = st = en = 0;
    Q[en++] = src;
    while(st < en) {
        u = Q[st++];
        for(i=fin[u]; i>=0; i=nxt[i]) {
            v = to[i];
            if(flow[i]  + eps < cap[i] && dist[v]==-1) {
                dist[v] = dist[u]+1;
                Q[en++] = v;
```

```
            }
        }
    }
    return dist[snk]!=-1;
}

long double dfs(int u, long double fl) {
    if(u==snk) return fl;
    long double df;

    for(int &e=pro[u], v; e>=0; e=nxt[e]) {
        v = to[e];
        if(flow[e]  + eps < cap[e] && dist[v]==dist[u]+1) {
            df = dfs(v, min(cap[e]-flow[e], fl));

            if(df> eps) {
                flow[e] += df;
                flow[e^1] -= df;
                return df;
            }
        }
    }
    return 0;
}

long double dinitz() {
    long double ret = 0, df;

    while(bfs()) {
        for(int i=1; i<=nNode; i++) pro[i] = fin[i];
        while(true) {
            df = dfs(src, INF);
            if(fabs(df) > eps) ret += df;
            else break;
        }
    }
    return ret;
}
```

4.11 min cost max flow

```

// Works only on directed Graph
// *** 1 based indexing

#define MAXN    203
#define MAXE    100000
const int INF = 0x7f7f7f7f;

int src, snk, nNode, nEdge;
int fin[MAXN + 5], pre[MAXN + 5], dist[MAXN + 5];
int cap[2*MAXE+5], cost[2*MAXE+5], Next[2*MAXE+5], to[2*MAXE+5], from[2*MAXE
```

```
    +5];

inline void init(int _src, int _snk, int nodes) {
    memset(fin, -1, sizeof(fin));
    nNode = nodes, nEdge = 0;
    src = _src, snk = _snk;
}

inline void addEdge(int u, int v, int _cost, int _cap) {
    from[nEdge] = u, to[nEdge] = v, cap[nEdge] = _cap, cost[nEdge] = _cost;
    Next[nEdge] = fin[u], fin[u] = nEdge++;
    from[nEdge] = v, to[nEdge] = u, cap[nEdge] = 0, cost[nEdge] = -(_cost);
    Next[nEdge] = fin[v], fin[v] = nEdge++;
    assert(nEdge <= 2*MAXE);
}

bool bellman() {
    int iter, u, v, i;
    bool flag = true;
    memset(dist, 0x7f, sizeof(dist));
    memset(pre, -1, sizeof(pre));
    dist[src] = 0;
    for(iter = 1; iter < nNode && flag; iter++) {
        flag = false;
        for(u = 1; u <= nNode; u++) {
            for(i = fin[u]; i >= 0; i = Next[i]) {
                v = to[i];
                if(cap[i] && dist[v] > dist[u] + cost[i]) {
                    dist[v] = dist[u] + cost[i];
                    pre[v] = i;
                    flag = true;
                }
            }
        }
    }
    return (dist[snk] < INF);
}

int mcmf(int &fcost) {
    int netflow, i, bot, u;
    netflow = fcost = 0;
    while(bellman()) {
        bot = INF;
        for(u = pre[snk]; u >= 0; u = pre[from[u]]) bot = min(bot, cap[u]);
        for(u = pre[snk]; u >= 0; u = pre[from[u]]) {
            cap[u] -= bot;
            cap[u^1] += bot;
            fcost += bot * cost[u];
        }
        netflow += bot;
    }
```

```
    return netflow;
}
```

4.12 mst

```

// Minimum Spanning Tree
VI edge[MAX+10];
int par[MAX+10];  //n ->number of nodes
VI mst_tree[MAX+10], mst_cost[MAX+10];

struct Eg{
    int u, v, cost;
    Eg(){};
    Eg(int a, int b, int c) {u = a, v = b, cost = c; }
};
vector<Eg> E; // contains the edges

bool cmp(Eg a, Eg b) {return a.cost < b.cost; }

int find_parent(int idx)
{
    return par[idx] == idx? idx:par[idx] = find_parent(par[idx]);
}

int mst(int n)
{
    int i, ret = 0;
    Eg e;
    for(i = 0; i <= n; i++) par[i] = i;

    sort(E.begin(), E.end(), cmp);
    for(i = 0; i < (int) E.sz; i++)
    {
        e = E[i];
        if(find_parent(e.u) == find_parent(e.v)) continue;
        else
        {
            par[find_parent(e.u)] = find_parent(e.v), ret += e.cost;
            mst_tree[e.u].pb(e.v);
            mst_tree[e.v].pb(e.u);
            mst_cost[e.u].pb(e.cost);
            mst_cost[e.v].pb(e.cost);
        }
    }
    return ret;
}
```

4.13 sparse table

```

//Sparse Table
```

```
#define MAXLG   17
#define MAXN    10000
int Tab[MAXLG+5][MAXN+11], par[MAXN+11], lev[MAXN+11], stp; // par->parent, lev = level;

void init_st(int n)
{
    int idx;

    for(idx = 1; idx <= n; idx++)
        Tab[0][idx] = par[idx];

    for(stp = 1; (1 << stp) < n; stp++)
        for(idx = 1; idx <= n; idx++)
            if(Tab[stp-1][idx] == -1) Tab[stp][idx] = -1;
            else Tab[stp][idx] = Tab[stp-1][Tab[stp-1][idx]];
    stp--;
}

int go(int cur, int gap, int pos)
{
    if(!gap) return cur;
    if(gap&1) return go(Tab[pos][cur], gap/2, pos+1);
    return go(cur, gap/2, pos+1);
}

int LCA(int u, int v)
{
    if(lev[u] > lev[v]) swap(u,v);
    v = go(v, lev[v]-lev[u], 0);
    if(u == v) return v;

    int i;
    for(i = stp; i >= 0; i--)
        if(Tab[i][u] != Tab[i][v])
            u = Tab[i][u], v = Tab[i][v];

    assert(Tab[0][u] > 0);
    return Tab[0][u];
}
```

4.14 two sat

```
/// 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();
        }
    }

    inline void add_edge(int a, int b){
        adj[neg(a)].push_back(b);
        rev[b].push_back(neg(a));
    }


    /// Adds constraint on a and b so that either a or b (or both) must be
       true
    /// Negative values means not

    /// For example:
    /// either a or b true: add_constraint(a, b)
    /// either a or b false: add_constraint(-a, -b)
    /// either a or b true but not both: add_constraint(a, b), add_constraint
       (-a, -b)

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

        add_edge(a, b);
        add_edge(b, a);
    }

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


    /// a or b (-x implies !x)
    inline void add_or(int a, int b){
        add_constraint(a, b);
    }

    /// a xor b (-x implies !x)
    inline void add_xor(int a, int b){
        add_constraint(a, b);
        add_constraint(-a, -b);
```

```
59      }
60
61      /// a and b (-x implies !x)
62      inline void add_and(int a, int b){
63          add_constraint(a, b);
64          add_constraint(a, -b);
65          add_constraint(-a, b);
66      }
67
68      /// force variable a to be true (if a is negative, force !a to be true)
69      inline void force_true(int a){
70          if (a < 0) a = n - a;
71          adj[neg(a)].push_back(a);
72          rev[a].push_back(neg(a));
73      }
74
75      /// force variable a to be false (if a is negative, force !a to be false)
76      inline void force_false(int a){
77          if (a < 0) a = n - a;
78          adj[a].push_back(neg(a));
79          rev[neg(a)].push_back(a);
80      }
81
82      inline void topsort(int i){
83          visited[i] = true;
84          int j, x, len = rev[i].size();
85
86          for (j = 0; j < len; j++){
87              x = rev[i][j];
88              if (!visited[x]) topsort(x);
89          }
90          dfs_t[i] = ++l;
91      }
92
93      inline void dfs(int i, int p){
94          parent[i] = p;
95          visited[i] = true;
96          int j, x, len = adj[i].size();
97
98          for (j = 0; j < len; j++){
99              x = adj[i][j];
100             if (!visited[x]) dfs(x, p);
101         }
102     }
103
104     void build(){
105         int i, x;
106         clr(visited);
107         for (i = m, l = 0; i >= 1; i--){
108             if (!visited[i]){
109                 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;
            }
        }

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

5 Matrices

5.1 Gauss E Maxx

```

//Gaussian Elimination
vector<double> res;
vector< vector< double> > mat;
//0-Based Indexing
int gauss (vector < vector<double> > a, vector<double> & ans)
{
    int n = (int) a.size(); // number of rows
    int m = (int) a[0].size() - 1; // number of columns - 1

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col)
```

```
    {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row)
            {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i)
    {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

5.2 Gauss Number Of Spanning tree of a weighted simple tree

```

#include<bits/stdc++.h>
using namespace std;
#define D(x)    cout << #x " = " << (x) << endl
#define MAX     100
typedef long long int LL;
/*
    way[i][j] = number of ways to create the j-th vertex module the i-th prime
    No loops / multi - edge
    Must be undirected
```

```
    MUST CALL CLEAR BEFORE CALLING RESIZE on A
*/

int mod[] = {3, 10337}, way[2][MAX+5], edge[MAX+5][MAX+5], n, nxt_edge[MAX+5][MAX+5], nxt_way[2][MAX+5], par[MAX+5], counter[MAX+5][MAX+5], nxt_counter[MAX+5][MAX+5];
vector<int> wlist, nodeList[MAX+5];
bool vis[MAX+5];

void dfs(int idx, int cmpNo, int cost)
{
    if(vis[idx]) return;

    par[idx] = cmpNo;
    vis[idx] = true;
    nodeList[cmpNo].push_back(idx);

    for(int i = 1; i <= n; i++)
        if(edge[idx][i] == cost)
            dfs(i, cmpNo, cost);
}

LL mul(LL u, LL v, LL m)
{
    if(u >= m) u %= m;
    if(v >= m) v %= m;
    LL ret = u * v;
    if(ret >= m) return ret % m;
    return ret;
}

LL add(LL u, LL v, LL m)
{
    return (u + v) % m;
}

LL sub(LL u, LL v, LL m)
{
    LL ret = (u - v) % m;
    if(ret < 0) ret += m;
    return ret;
}

LL ip(LL a, LL p, LL m)
{
    if(!p) return 1;
    if(p & 1) return mul(a, ip(a, p - 1, m), m);
    LL ret = ip(a, p/2, m);
    return mul(ret, ret, m);
}

```

```
LL mod_inv(LL v, LL m){return ip(v, m - 2, m);}

int gauss(vector < vector < int > > A,  int m){
    int eqn = A.size();
    int var = A.back().size() - 1, i, j;
    vector<int> where;
    where.resize(var, -1);
    LL c, d = 1, ret = 1;


    for(int clm = 0, row = 0; clm < var && row < eqn; clm++){
        if(!A[row][clm])
            for(i = row + 1; i < eqn; i++)
                if(A[i][clm]) {
                    for(j = 0; j <= var; j++)
                        swap(A[i][j], A[row][j]);
                    d = (-d) % m;
                    if(d < 0) d += m;

                    break;
                }

        if(!A[row][clm]) {continue;}
        where[clm] = row;

        for(c = mod_inv(A[row][clm], m), i = 0, d = mul(d, c, m); i <= var; i
            ++) {
            A[row][i] = mul( A[row][i] , c , m);
        }

        for(i = 0; i < eqn; i++)
            if(i == row) continue;
            else
                for(c = A[i][clm], j = 0; j <= var; j++) {
                        A[i][j] = sub(A[i][j] , mul(A[row][j] , c, m), m);
                    }
        row++;
    }

    for(i = 0; i < eqn; i++)
        ret = mul(ret, A[i][i], m);

    d = mod_inv(d, m);
    return mul(ret, d, m);
}

vector< vector<int> > A, B;
int deg[MAX+5];

LL kirchoff(vector<int> node, int cost, int m)
{
```

```
110     if(node.size() == 1) return 1;
111
112     A.clear();
113     A.resize(node.size());
114     memset(deg, 0, sizeof(deg));
115
116     for(int i = 0; i < (int) node.size(); i++) A[i].resize(node.size() + 1);
117
118     for(int i = 0; i < (int) node.size(); i++)
119         for(int j = i + 1; j < (int) node.size(); j++){
120             int u = node[i];
121             int v = node[j];
122
123             if(edge[u][v] == cost){
124                 deg[i] = add(deg[i], counter[u][v], m), deg[j] = add(deg[j],
                        counter[u][v], m);
125                 A[i][j] = A[j][i] = (-counter[u][v] % m + m) % m;
126             }
127         }
128
129     for(int i = 0; i < (int) node.size(); i++)
130         A[i][i] = deg[i];
131
132
133     B.clear();
134     B.resize(node.size() - 1);
135     for(int i = 0; i < (int) B.size(); i++)
136     {
137         B[i].resize((int) A[i].size() - 1);
138         for(int j = 0; j < (int) B[i].size(); j++)
139             B[i][j] = A[i][j];
140     }
141
142     return gauss(B, m);
143 }
144
145 void _merge(int cost)
146 {
147     int i, p, w, j, cmpNo = 0;
148     memset(vis, 0, sizeof(vis));
149
150     for(i = 1; i <= n; i++)
151         if(vis[i] == false){
152             cmpNo++;
153             nodeList[cmpNo].clear();
154             dfs(i, cmpNo, cost);
155
156             for(p = 0; p < 2; p++)
157             {
158                 w = kirchoff(nodeList[cmpNo], cost, mod[p]);
159
```

```
                for(auto x : nodeList[cmpNo]) w = (w * (LL) way[p][x]) % mod[p
                    ];
                nxt_way[p][cmpNo] = w;
            }
        }

    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)
            nxt_edge[i][j] = numeric_limits<int>::max();

    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)
            if(edge[i][j] != numeric_limits<int>::max()){
                int u = par[i];
                int v = par[j];

                if(u == v) continue;

                if(edge[i][j] < nxt_edge[u][v]){
                    nxt_edge[u][v] = edge[i][j];
                    nxt_counter[u][v] = counter[i][j];
                }
                else if(edge[i][j] == nxt_edge[u][v])
                {
                    nxt_counter[u][v] += counter[i][j];
                }
            }

    n = cmpNo;
    for(p = 0; p < 2; p++)
        for(i = 1; i <= n; i++)
            way[p][i] = nxt_way[p][i];

    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)
        {
            edge[i][j] = nxt_edge[i][j];
            counter[i][j] = nxt_counter[i][j];
        }
}

vector<int> prime;
vector<int> rm;

int crt()
{
    LL M = 1, ret = 0, b, c;
    for(auto x : prime)
        M = M * x;

    for(int i = 0; i < (int) prime.size(); i++){
```

```
        b = mod_inv( M / prime[i], prime[i]);
        c = (b * rm[i]) % M;
        c = (c * M/prime[i]) % M;
        ret = (ret + c) % M;
    }
    return ret;
}

int main()
{
    //freopen("in.txt", "r", stdin);
    //freopen("out.txt", "w", stdout);


    int p, m, u, v, w, i, j, itr;

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

    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)
            edge[i][j] = numeric_limits<int>::max();

    for(p = 0; p < 2; p++)
        for(i = 1; i <= n; i++)
            way[p][i] = 1;


    for(i = 1; i <= m; i++)
    {
        scanf("%d %d %d", &u, &v, &w);
        edge[u][v] = edge[v][u] = w;
        counter[u][v] = counter[v][u] = 1;

        wlist.push_back(w);
    }

    sort(wlist.begin(), wlist.end());
    wlist.erase(unique(wlist.begin(), wlist.end()), wlist.end());

    itr = 0;
    while(n != 1){
        assert(itr < wlist.size());
        w = wlist[itr++];
        _merge(w);
    }

    for(p = 0; p < 2; p++) {
        prime.push_back(mod[p]);
        rm.push_back(way[p][1]);
    }

```

```
261     printf("%d\n", crt());
262     return 0;
263 }
```

5.3 Gauss for BigInteger And Fraction

```
1
2  /*
3      Gauss for BigInteger + Fraction
4  */
5
6  import java.math.BigInteger;
7  import java.util.Scanner;
8
9  public class Main {
10
11     public static void main(String args[])
12     {
13
14         int i, t, cs, lim;
15         String s;
16         Scanner sf = new Scanner(System.in);
17
18         t = sf.nextInt();
19         for(cs = 1; cs <= t; cs++){
20             lim = sf.nextInt();
21             s = sf.next();
22
23             KMP k = new KMP(s);
24             k.failure_function();
25
26             Matrix M = new Matrix(s.length() + 1, s.length() + 1);
27
28             for(i = 0; i < s.length(); i++){
29
30                 M.mat[i][i] = new Fraction(-1, 1);
31                 M.mat[i][s.length() + 1] = new Fraction(-1, 1);
32
33                 for(int x = 1; x <= lim; x++){
34                     if(s.charAt(i) == (char) ('A' + x - 1)){
35                         M.mat[i][i+1] = M.mat[i][i + 1].add(new Fraction(1,
                               lim));
36                     }
37                     else{
38                         boolean matched = false;
39                         int npos = i;
40                         while(npos != 0){
41                             npos = k.F[npos];
42
43                             if(s.charAt(npos) == (char) ('A' + x - 1)){
44                                 M.mat[i][npos + 1] = M.mat[i][npos + 1].add(
                                       new Fraction(1, lim));
```

```
                                    matched = true;
                                    break;
                                }
                            }

                            if(matched == false){
                                M.mat[i][0] = M.mat[i][0].add(new Fraction(1, lim
                                    ));
                            }
                        }
                    }
                }

                M.mat[s.length()][s.length()] = new Fraction(1, 1);
                M.mat[s.length()][s.length() + 1] = new Fraction(0, 1);

                M.elim();
                System.out.println("Case " + cs + ":");
                System.out.println(M.ret[0].up);
                if(cs != t) System.out.println("");
            }
        }
    }

class KMP{
    String str;
    int F[];

    KMP(String s){
        str = s;
        F = new int[str.length() + 5];
    }

    void failure_function(){
        int len = str.length(), idx, i;

        F[0] = F[1] = 0;
        for(idx = 2; idx <= len; idx++)
        {
            i = F[idx  - 1];
            while(true)
            {
                if(str.charAt(i) == str.charAt(idx - 1)) {
                    F[idx] = i + 1;
                    break;
                }
                else if(i != 0) i = F[i];
                else{
                    F[idx] = 0;
                    break;
                }
```

```
            }
        }
    }
}

class Fraction{
    BigInteger up, dwn;

    void reduce()
    {
        BigInteger g = up.gcd(dwn);
        up = up.divide(g);
        dwn = dwn.divide(g);

        if(up.signum() == -1 && dwn.signum() == -1){
            up = up.negate();
            dwn = dwn.negate();
        }
    }

    Fraction(){
        up = BigInteger.valueOf(0);
        dwn = BigInteger.valueOf(1);
        reduce();
    }

    Fraction(int u, int d){
        up = BigInteger.valueOf(u);
        dwn = BigInteger.valueOf(d);
        reduce();
    }

    Fraction(BigInteger u, BigInteger v){
        up = u;
        dwn = v;
        reduce();
    }

    Fraction add(Fraction ano){
        Fraction F = new Fraction( up.multiply(ano.dwn).add(ano.up.multiply(
            dwn))  , dwn.multiply(ano.dwn) );
        return F;
    }

    Fraction sub(Fraction ano){
        Fraction F = this;
        F = F.add(new Fraction(ano.up.negate(), ano.dwn));
        return F;
    }

    Fraction multiply(Fraction ano){
```

```
145         Fraction F = new Fraction( up.multiply(ano.up), dwn.multiply(ano.dwn))
                ;
146         return F;
147     }
148
149     Fraction divide(Fraction ano){
150         Fraction F = new Fraction( up.multiply(ano.dwn), dwn.multiply(ano.up))
                ;
151         return F;
152     }
153
154     boolean equals(Fraction ano){
155         return up.multiply(ano.dwn).equals(ano.up.multiply(dwn));
156     }
157
158     void copy(Fraction ano){
159         up = ano.up;
160         dwn = ano.dwn;
161     }
162 }
163
164 class Matrix{
165     int neq, nvar;
166     Fraction mat[][];
167     Fraction ret[];
168
169
170     Matrix(int r, int c){
171         neq = r;
172         nvar = c;
173         mat = new Fraction[r][c + 1];
174         ret = new Fraction[c];
175
176         for(int i = 0; i < r; i++)
177             for(int j = 0; j <= c; j++)
178                 mat[i][j] = new Fraction(0, 1);
179
180         for(int i = 0; i < nvar; i++)
181             ret[i] = new Fraction(0, 1);
182     }
183
184     int elim(){
185         int free_var = 0;
186         int eqn = neq;
187         int var = nvar;
188         int where[] = new int[nvar];
189         for(int i = 0; i < nvar; i++) where[i] = -1;
190         Fraction zero = new Fraction(0, 1);
191         Fraction c = new Fraction(0, 1);
192
193         int row = 0;
```

```
194         for(int clm = 0; clm < var && row < eqn; clm++)
195         {
196             if(mat[row][clm].equals(zero)){
197                 for(int i = row + 1; i < eqn; i++)
198                 {
199                     if(mat[row][clm].equals(zero) == false){
200                         for(int j = 0; j <= var; j++){
201                             Fraction tmp = new Fraction();
202                             tmp.copy(mat[i][j]);
203                             mat[i][j].copy(mat[row][j]);
204                             mat[row][j].copy(tmp);
205                         }
206                     }
207                 }
208             }
209 
210             if(mat[row][clm].equals(zero)) continue;
211             where[clm] = row;
212 
213             c.copy(mat[row][clm]);
214             for(int  i = 0; i <= var; i++){
215                 mat[row][i] = mat[row][i].divide(c);
216             }
217 
218             for(int i = 0; i < eqn; i++)
219                 if(i == row) continue;
220                 else{
221                     c.copy(mat[i][clm]);
222                     for(int j = 0; j <= var; j++){
223                         Fraction tmp = new Fraction(0, 1);
224                         tmp.copy(mat[row][j]);
225                         tmp = tmp.multiply(c);
226                         mat[i][j] = mat[i][j].sub(tmp);
227                     }
228                 }
229             row++;
230         }
231 
232         for(int i = 0; i < var; i++) ret[i] = new Fraction(0, 1);
233         for(int i = 0; i < var; i++){
234             if(where[i] != -1) ret[i].copy(mat[where[i]][var]);
235             else free_var++;
236         }
237 
238         for(int i = 0; i < eqn; i++){
239             Fraction sum = new Fraction(0, 1);
240             for(int j = 0; j < var; j++){
241                 Fraction tmp = new Fraction(0, 1);
242                 tmp.copy(mat[i][j]);
243                 tmp = tmp.multiply(ret[j]);
244                 sum = sum.add(tmp);
```

```
            }
            if(sum.equals(mat[i][var]) == false) return 0;
        }

        if(free_var != 0) return 1000000000;
        return 1;
    }
}
```

5.4 Gauss for doubles

```

/*
    COMPLEXITY: min(eqn, var) * eqn * var
    MUST CALL CLEAR BEFORE CALLING RESIZE on A
*/

int gauss(vector < vector < double > > A, vector < double > &ret){
    /*
        0-based indexing
        n = number of variables
        m = number of equations

        a_11 a_12 a_13 ..... a_1n | e_1
        a_21 a_22 a_23 ..... a_2n | e_2
        ...
        ...
        ...
        a_m1 a_m2 a_m3 ..... a_mn | e_m

    */

    if(!A.size()) return 1;
    int eqn = A.size();
    int var = A.back().size() - 1, i, j;
    int free_var = 0;
    vector<int> where;
    where.assign(var, -1);
    double c;


    for(int clm = 0, row = 0; clm < var && row < eqn; clm++){

        /*
            Iterating over the variables
                - if (the i-th column is full of 0) then the i-th variable is free
                - else do operations to make sure that A[row][clm] = 1 and A[row'][clm] = 0 when row' != row
        */

        if(abs(A[row][clm]) < eps)
```

```
            for(i = row + 1; i < eqn; i++)
                if(abs(A[i][clm]) > eps) {
                    for(j = 0; j <= var; j++)
                        swap(A[i][j], A[row][j]);
                    break;
                }

        if(abs(A[row][clm]) < eps) {continue;}
        where[clm] = row;

        for(i = 0; i < row; i++) assert(abs(A[row][i]) < eps);
        for(c = A[row][clm], i = 0; i <= var; i++) A[row][i] /= c;

        for(i = 0; i < eqn; i++)
            if(i == row) continue;
            else for(c = A[i][clm], j = 0; j <= var; j++) A[i][j] = A[i][j] -
                A[row][j] * c;

        row++;
    }

    ret.assign(var, 0.0); // We MUST let the free variables to take the value
        0.
    for(i = 0; i < var; i++)
        if(where[i] != -1) ret[i] = A[where[i]][var];
        else free_var++;


    for(i = 0; i < eqn; i++){
        double sum = 0;
        for(j = 0; j < var; j++)
            sum += A[i][j] * ret[j];

        if(fabs(sum - A[i][var]) > eps) return 0;
    }

    if(free_var) return INF;
    return 1;
}
```

5.5 Gauss for modular eqn

```

/*
    COMPLEXITY: min(eqn, var) * eqn * var
    The MOD must be a prime.
    MUST CALL CLEAR BEFORE CALLING RESIZE on A
*/

int gauss(vector < vector < int > > A, vector < int > &ret){

    if(!A.size()) return 1;
```

```
    int eqn = A.size();
    int var = A.back().size() - 1, i, j;
    int free_var = 0;
    vector<int> where;
    where.resize(var, -1);
    LL c;


    for(int clm = 0, row = 0; clm < var && row < eqn; clm++){
        if(!A[row][clm])
            for(i = row + 1; i < eqn; i++)
                if(A[i][clm]) {
                    for(j = 0; j <= var; j++)
                        swap(A[i][j], A[row][j]);
                    break;
                }

        if(!A[row][clm]) {continue;}
        where[clm] = row;

        for(i = 0; i < row; i++) assert(!A[row][i]);
        for(c = mod_inv(A[row][clm]), i = 0; i <= var; i++) A[row][i] = mul( A
            [row][i] , c );

        for(i = 0; i < eqn; i++)
            if(i == row) continue;
            else
                for(c = A[i][clm], j = 0; j <= var; j++) {
                        A[i][j] = sub(A[i][j] , mul(A[row][j] , c));
                    }
        row++;
    }

    ret.assign(var, 0);
    for(i = 0; i < var; i++)
       if(where[i] != -1) ret[i] = A[where[i]][var];
       else free_var++;

    for(i = 0; i < eqn; i++){
        int sum = 0;
        for(j = 0; j < var; j++)
            sum = add(sum ,  mul(A[i][j] , ret[j]));
        if(sum != A[i][var]) return 0;
    }

    if(free_var) return INF;
    return 1;
}
```

5.6 Gauss maximum xor subset

```

```

```cpp
#include<bits/stdc++.h>
using namespace std;
#define D(x)    cout << #x " = " << (x) << endl
typedef long long int LL;

const int N = 101;
/*
    COMPLEXITY: min(eqn, var) * eqn * var

    Size of the bitsets HAS TO BE constant.
    So we need to use the max value.

    Code works for signed 64 bit non-negative integer
    MUST CALL CLEAR BEFORE CALLING RESIZE on A
*/

const int INF = numeric_limits<int>::max();

int gauss(vector < bitset < N > > A, vector < bool > &ret, int nVar){

    if(!A.size()) return 1;
    int eqn = A.size();
    int var = nVar, i, j;
    int free_var = 0;
    bool c;
    vector<int> where;
    where.assign(var, -1);


    for(int clm = 0, row = 0; clm < var && row < eqn; clm++){
        if(!A[row][clm])
            for(i = row + 1; i < eqn; i++)
                if(A[i][clm]) {
                    for(j = 0; j <= var; j++)
                    {
                        bool tmp = A[i][j];
                        A[i][j] = A[row][j];
                        A[row][j] = tmp;
                    }
                    break;
                }

        if(!A[row][clm]) {continue;}

        where[clm] = row;
        for(i = 0; i < clm; i++) assert(!A[row][i]);

        for(i = 0; i < eqn; i++)
            if(i == row) continue;
            else{
                /*
```

```
                    If the input file consists of multiple test case tweak here
                    As the size of bitset may be a lot bigger, think about looping replacing the xor operation
                */
                c = A[i][clm];
                if(c) A[i] = A[i] ^ A[row];
            }

        row++;
    }

    ret.assign(var, 0);

    for(i = 0; i < var; i++)
        if(where[i] != -1) ret[i] = A[where[i]][var];
        else free_var++;

    for(i = 0; i < eqn; i++){
        bool sum = 0;
        for(j = 0; j < var; j++)
            sum ^= A[i][j] * ret[j];

        if(sum != A[i][var]) return 0;
    }

    if(free_var) return INF;
    return 1;
}

char str[66];
vector < bitset < N > > A;
vector < bool > tmp;

bool can_make(vector<LL> &input, int idx){ // Is it possible to make the prefix from idx to 62?
    A.clear();
    A.resize(62 - idx + 1);

    for(int pos = 62, eqn = 0; pos >= idx; pos--, eqn++){
        for(int i = 0; i < (int) input.size(); i++){
            if(input[i] & (1LL << pos)) A[eqn][i] = true;
        }

        A[eqn][input.size()] = (str[pos] == '1') ? true : false;
    }

    if(!gauss(A, tmp, input.size())) return false;
    return true;
}

```

```cpp
LL max_xor_subset(vector <LL> &input)
{
    LL ret = 0;
    memset(str, 0, sizeof(str));
    for(int pos = 62; pos >= 0; pos--){
        str[pos] = '1';
        if(can_make(input, pos) == false) str[pos] = '0';
        else ret += (1LL << pos);
    }

    return ret;
}

int main()
{
    //freopen("in.txt", "r", stdin);

    int i, n, t, cs;
    vector<LL> seq;
    LL v;

    scanf("%d", &t);
    for(cs = 1; cs <= t; cs++)
    {
        seq.clear();

        scanf("%d", &n);
        for(i = 1; i <= n; i++)
        {
            scanf("%lld", &v);
            seq.push_back(v);
        }

        printf("Case %d: %lld\n", cs, max_xor_subset(seq));
    }
    return 0;
}
```

5.7 mat exp

```cpp

/*
    Matrix Exponentiation :: FIX THE DIM FOR EVERY MATRIX
    ****** 1 based indexing ******
*/
#define DIM 5
#define EXPM 1000000007
struct matrix{
    int mat[DIM+5][DIM+5], dim;
    matrix(){}
    matrix(int d, int x = 1){                    ///* USE WITH CAUTION *///
        int i,j;
```

```
        dim = d;
        for(i = 1; i <= dim; i++)
            for(j = 1; j <= dim; j++)
                mat[i][j] = (i == j) ? x : 0;
    }
    matrix operator * (const matrix &r){
        int i, j, k;
        matrix ret = matrix(dim);
        assert(dim == r.dim);
        for(i = 1; i <= dim; i++)
            for(j = 1; j <= dim; j++)
            {
                ret.mat[i][j] = 0;
                for(k = 1; k <= ret.dim; k++)
                    ret.mat[i][j] = (ret.mat[i][j] + ((LL)mat[i][k]*r.mat[k][j
                        ])%EXPM)%EXPM;
            }
        return ret;
    }
};

matrix expo(matrix &in, LL p)
{
    matrix ret = matrix(in.dim), aux = in;
    while(p)
    {
        if(p&1) ret = ret*aux;
        aux = aux*aux;
        p>>=1;
    }
    return ret;
}
```

6 Number Theory

6.1 CRT SOLVER

```
/**
    A CRT solver which works even when moduli are not pairwise coprime
    1. Add equations using addEquation() method
    2. Call solve() to get {x, N} pair, where x is the unique solution modulo
       N.
    Assumptions:
        1. LCM of all mods will fit into long long.
*/
class ChineseRemainderTheorem {
    typedef long long vlong;
    typedef pair<vlong,vlong> pll;

    /** CRT Equations stored as pairs of vector. See addEqation()*/
    vector<pll> equations;

```

```
public:
    void clear() {
        equations.clear();
    }

    /** Add equation of the form x = r (mod m)*/
    void addEquation( vlong r, vlong m ) {
        equations.push_back({r, m});
    }
    pll solve() {
        if (equations.size() == 0) return {-1,-1}; /// No equations to solve

        vlong a1 = equations[0].first;
        vlong m1 = equations[0].second;
        a1 %= m1;
        /** Initially x = a_0 (mod m_0)*/

        /** Merge the solution with remaining equations */
        for ( int i = 1; i < equations.size(); i++ ) {
            vlong a2 = equations[i].first;
            vlong m2 = equations[i].second;

            vlong g = __gcd(m1, m2);
            if ( a1 % g != a2 % g ) return {-1,-1}; /// Conflict in equations

            /** Merge the two equations*/
            vlong p, q;
            ext_gcd(m1/g, m2/g, &p, &q);

            vlong mod = m1 / g * m2;
            vlong x = ( (__int128)a1 * (m2/g) % mod *q % mod + (__int128)a2 *
                (m1/g) % mod * p % mod ) % mod;

            /** Merged equation*/
            a1 = x;
            if ( a1 < 0 ) a1 += mod;
            m1 = mod;
        }
        return {a1, m1};
    }
};
```

6.2 FFTW

```

const double PI = acos(-1.0);

#define SGT_MAX 2097152 /// 2 * Smallest power of 2 greater than MAXN, 2^18
    when MAXN = 10^5
typedef complex <double> complx; /// Replace double with long double if more
    precision is required
complx dp[SGT_MAX >> 1], P1[SGT_MAX], P2[SGT_MAX];
```

```
7
8  inline long long round_half_even(double x){
9      long long res = abs(x) + 0.5;
10     if (x < 0) res = -res;
11     return res;
12 }
13
14 void FFT(complx *ar, int n, int inv){
15     int i, j, l, len, len2;
16     const double p = 4.0 * inv * acos(0.0);
17
18     for (i = 1, j = 0; i < n; i++){
19         for (l = n >> 1; j >= l; l >>= 1) j -= l;
20         j += l;
21         if (i < j) swap(ar[i], ar[j]);
22     }
23
24     for (len = 2; len <= n; len <<= 1){
25         len2 = len >> 1;
26         double theta = p / len;
27         complx mul(cos(theta), sin(theta));
28
29         dp[0] = complx(1, 0);
30         for (i = 1; i < len2; i++) dp[i] = (dp[i - 1] * mul);
31
32         for (i = 0; i < n; i += len){
33             complx t, *pu = ar + i, *pv = ar + i + len2, *pend = ar + i + len2
                   , *pw = dp;
34             for (; pu != pend; pu++, pv++, pw++){
35                 t = (*pv) * (*pw);
36                 *pv = *pu - t;
37                 *pu += t;
38             }
39         }
40     }
41
42     if (inv == -1){
43         for (i = 0; i < n; i++) ar[i] /= n;
44     }
45 }
46
47 int multiply(int a, complx* A, int b, complx* B){
48     int i, n, m;
49     n = a + b - 1;
50     m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) == 1));
51
52     for (i = a; i < m; i++) A[i] = 0;
53     for (i = b; i < m; i++) B[i] = 0;
54     FFT(A, m, 1), FFT(B, m, 1);
55     for (i = 0; i < m; i++) A[i] = A[i] * B[i];
56     FFT(A, m, -1);
```

```
    return m;
}

void multiply (const vector<int> & a, const vector<int> & b, vector<int> & res) {
    for(int i = 0; i < (int) a.size(); i++) P1[i] = complx(a[i], 0);
    for(int i = 0; i < (int) b.size(); i++) P2[i] = complx(b[i], 0);
    int degree = multiply(a.size(), P1, b.size(), P2);
    res.resize(degree);
    for(int i = 0; i < degree; i++) res[i] = round_half_even(P1[i].real());
    while(!res.back()) res.pop_back();
}
```

6.3 FFT long

```

#define SGT_MAX 1048576 /// 2 * MAX at least
/// Change long double to double if not required

namespace fft{
    int len, last = -1, step = 0, rev[SGT_MAX];
    long long C[SGT_MAX], D[SGT_MAX], P[SGT_MAX], Q[SGT_MAX];

    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[SGT_MAX], v[SGT_MAX], f[SGT_MAX], g[SGT_MAX], dp[SGT_MAX], inv[SGT_MAX];

    inline long long round_half_even(long double x){
        long long res = abs(x) + 0.5;
        if (x < 0) res = -res;
        return res;
    }

    /// 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) < SGT_MAX; 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] = round_half_even(f[i].real / (long double)len);
        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] = round_half_even(f[i].real);
        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;
    }
```

```
182
183     /// Multiplies two polynomials where intermediate and final values fits in
            long long
184     /// Does not work for negative values because it inherently uses
            mod_multiply()
185
186     int long_multiply(int a, long long* A, int b, long long* B){
187         int mod1 = 1.5e9;
188         int mod2 = mod1 + 1;
189         for (int i = 0; i < a; i++) C[i] = A[i];
190         for (int i = 0; i < b; i++) D[i] = B[i];
191
192         mod_multiply(a, A, b, B, mod1);
193         mod_multiply(a, C, b, D, mod2);
194         for (int i = 0; i < len; i++){
195             A[i] = A[i] + (C[i] - A[i] + (long long)mod2) * (long long)mod1 %
                    mod2 * mod1;
196         }
197         return a + b - 1;
198     }
199
200     int build_convolution(int n, long long* A, long long* B){
201         int i, m, d = 0;
202         for (i = 0; i < n; i++) Q[i] = Q[i + n] = B[i];
203         for (i = 0; i < n; i++) P[i] = A[i], P[i + n] = 0;
204         n *= 2, m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) == 1)
                );
205         for (i = n; i < m; i++) P[i] = Q[i] = 0;
206         return n;
207     }
208
209     /***
210         Computes the circular convolution of A and B, denoted A * B, in C
211         A and B must be of equal size, if not normalize before calling
                function
212         Example to demonstrate convolution for n = 5:
213
214         c0 = a0b0 + a1b4 + a2b3 + a3b2 + a4b1
215         c1 = a0b1 + a1b0 + a2b4 + a3b3 + a4b2
216         ...
217         c4 = a0b4 + a1b3 + a2b2 + a3b1 + a4b0
218
219
220         Note: If linear convolution is required, pad with zeros appropriately,
                as in multiplication
221
222     ***/
223
224     /// Returns the convolution of A and B in A
225     void convolution(int n, long long* A, long long* B){
226         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
    /// Does not work for negative values because it inherently uses
       mod_multiply()

    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 = "10011010011011101011101000"
    /// 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;
    }
}
```

6.4 FFT short

```
//FFT_Short

#define SGT_MAX 262148 /// 2 * Smallest power of 2 greater than MAXN, 2^18
   when MAXN = 10^5
typedef complex <double> complx; /// Replace double with long double if more
   precision is required
```

```
complx dp[SGT_MAX >> 1], P1[SGT_MAX], P2[SGT_MAX];

inline long long round_half_even(double x){
    long long res = abs(x) + 0.5;
    if (x < 0) res = -res;
    return res;
}

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);
```

```
55
56      FFT(A, m, 1), FFT(B, m, 1);
57      for (i = 0; i < m; i++) A[i] = A[i] * B[i];
58      FFT(A, m, -1);
59      for (i = 0; i < d && d; i++) A[i] = A[i + d];
60  }
61
62  int multiply(int a, complx* A, int b, complx* B){
63      int i, n, m;
64      n = a + b - 1;
65      m = 1 << (32 - __builtin_clz(n) - (__builtin_popcount(n) == 1));
66
67      for (i = a; i < m; i++) A[i] = 0;
68      for (i = b; i < m; i++) B[i] = 0;
69      FFT(A, m, 1), FFT(B, m, 1);
70      for (i = 0; i < m; i++) A[i] = A[i] * B[i];
71      FFT(A, m, -1);
72      return m;
73  }
74
75  /***
76      Computes the circular convolution of A and B, denoted A * B, in C
77      A and B must be of equal size, if not normalize before calling function
78      Example to demonstrate convolution for n = 5:
79
80      c0 = a0b0 + a1b4 + a2b3 + a3b2 + a4b1
81      c1 = a0b1 + a1b0 + a2b4 + a3b3 + a4b2
82      ...
83      c4 = a0b4 + a1b3 + a2b2 + a3b1 + a4b0
84
85
86      Note: If linear convolution is required, pad with zeros appropriately, as
            in multiplication
87
88  ***/
89
90  /// Returns the convolution of A and B in A
91
92  void convolution(int n, int* A, int* B){
93      for (int i = 0; i < n; i++) P1[i] = complx(A[i], 0);
94      for (int i = 0; i < n; i++) P2[i] = complx(B[i], 0);
95      convolution(n, P1, P2);
96      for (int i = 0; i < n; i++) A[i] = round_half_even(P1[i].real());
97  }
98
99  /// Multiplies two polynomials A and B and return the coefficients of their
        product in A
100 /// Function returns degree of the polynomial A * B
101
102 int multiply(int a, int* A, int b, int* B){
103     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] = round_half_even(P1[i].real());
    return degree;
}
```

6.5 FFT slow but short

```

///Fast Fourier Transformation
typedef complex<double> base;

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

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

    for (int len=2; len<=n; len<<=1) {
        double ang = 2*PI/len * (invert ? -1 : 1);
        base wlen (cos(ang), sin(ang));
        for (int i=0; i<n; i+=len) {
            base w (1);
            for (int j=0; j<len/2; ++j) {
                base u = a[i+j],  v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w *= wlen;
            }
        }
    }
    if (invert)
        for (int i=0; i<n; ++i)
            a[i] /= n;
}


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

    fft (fa, false),  fft (fb, false);
```

```
    for (size_t i=0; i<n; ++i)
        fa[i] *= fb[i];
    fft (fa, true);

    res.resize (n);
    for (size_t i=0; i<n; ++i)
        res[i] = int (fa[i].real() + 0.5);

    while(!res.back()) res.pop_back();
}
```

6.6 Grey code

```

// Returns the position of a grey code
// i-th grey code = i ^ (i / 2)

char bs[MAX+11]; //Keep the number here in binary representation
LL go(LL st, LL ed, int pos, bool isLeft = true)
{
    if(pos < 0) return 0;

    LL mid = (st+ed)/2;

    if(isLeft)
    {
        if(bs[pos] == '0') return go(st, mid, pos-1, true);
        else return (mid - st + 1) + go(mid+1, ed, pos-1, false);
    }
    else
    {
        if(bs[pos] == '1') return go(st, mid, pos-1, true);
        else return (mid - st + 1) + go(mid+1, ed, pos-1, false);
    }
}
LL getPos()
{
    int n = strlen(bs);

    reverse(bs, bs + n);
    LL ret = go(0, (1LL << n)-1, n-1);
    reverse(bs, bs+ n);
    return ret;
}
```

6.7 MILER RABIN

```
/**
    1. Works for n <= 2^64
    2. Uses at most 7 witnesses.
    3. Complexity: O( k log^3 n ), where k = 7
    Complexity should be reduced by log(n) factor since I used __int128
        instead of mul_mod(),
```

```
    but lets just consider it
    How to use it?
    You just call the only public method of the class, isPrime(n) and get boolean result.
*/

typedef long long vlong;

class MillerRabin {
    private:

    /** https://miller-rabin.appspot.com/ confirms that the following base covers 2^64**/
    vlong prime[7] = { 2, 325, 9375, 28178, 450775, 9780504, 1795265022 };
    int psize = 7;

    vlong bigmod ( __int128 a, __int128 p, vlong mod ) {
        __int128 x = a % mod, res = 1;
        while ( p ) {
            if ( p & 1 ) res = res * x % mod;
            x = x * x % mod;
            p >>= 1;
        }
        return res;
    }

    ///Witness to compositeness of n
    ///n - 1 = 2^s * d, where d is odd
    bool witness ( vlong a, vlong d, vlong s, vlong n ) {
        __int128 r = bigmod( a, d, n );
        if ( r == 1 || r == n - 1 ) return false;
        int i;
        for ( i = 0; i < s - 1; i++ ) {
            r = r * r % n;
            if ( r == 1 ) return true;
            if ( r == n - 1 ) return false;
        }
        return true;
    }

    public:
    bool isPrime ( vlong n ) {
        if ( n <= 1 ) return false;

        vlong p = n - 1, s = 0;
        while ( ! ( p & 1 ) ) {
            p /= 2;
            s++;
        }
        vlong d = p;
        p = n - 1;
```

```
        for ( int i = 0; i < psize && prime[i] < n; i++ ) {
            if ( witness ( prime[i], d, s, n ) ) return false;
        }
        return true;
    }
} millerRabin;
```

6.8 Mobius Function

```
//Mobius Function
void calc_mu(int n)
{
    int i, j;

    for(i = 1; i <= n; i++)
        mu[i] = mp(i,0);

    for(i = 2; i <= n; i++)
        if(!mu[i].ss)
            for(j = i; j <= n; j += i)
                mu[j].ff /= i, mu[j].ss++;

    for(i = 2; i <= n; i++)
        if(mu[i].ff != 1) mu[i].ff = 0;
        else if(mu[i].ss & 1) mu[i].ff = -1;
}
```

6.9 NEW Number of Solutions to an equation

```
// Minimizes |X| + |Y|
// Breaks tie with X <= Y
LL exEuclid(LL a, LL b, LL &x, LL &y)
{
    if(b == 0){
        x = 1;
        y = 0;
        return a;
    }

    LL nx, ny, g, r;
    g = exEuclid(b, a % b, nx, ny);
    x = ny;
    y = nx - a / b * ny;

    return g;
}

/*
    x0 -> Initial value
```

```
    dx -> Value which is going to be added with x0 in every step

    Returns the minimum number of steps needed to:
        1. make x0 > x
        2. get as close as possible to the value x maintaining the 1st
           constraint
*/

LL __lower_bound(LL x0, LL dx, LL x)
{
    LL d = x - x0;
    if(d > 0 && d % dx)
    {
        if(dx > 0) return d/dx + 1;
        return d/dx - 1;
    }
    return d/dx;
}

// Ax + By + C = 0
LL number_of_solution(LL A, LL B, LL C, LL x1, LL x2, LL y1, LL y2)
{
    LL g, x0, y0, dx, dy, stp, st1, st2, ed1, ed2, st, ed,x , y;

    if(A && B){
        g = exEuclid(A, B, x0, y0);
        if(C % g) return 0;
        else{
            x0 *= -C/g;
            y0 *= -C/g;
            dx = B/g;
            dy = A/g;

            if(dx < 0) dx *= -1, dy *= -1;

            stp = __lower_bound(x0, dx, -10000000000000LL);
            x0 += dx * stp;
            y0 -= dy * stp;

            st1 = __lower_bound(x0, dx, x1);
            ed1 = __lower_bound(x0, dx, x2 + 1) - 1;

            if(y0 > 0){
                ed2 = __lower_bound(y0, -dy, y1);
                st2 = __lower_bound(y0, -dy, y2 + 1) + 1;
            }
            else{
                st2 = __lower_bound(y0, -dy, y1);
                ed2 = __lower_bound(y0, -dy, y2 + 1) - 1;
            }

```

```
            st = max(st1, st2);
            ed = min(ed1, ed2);
            if(st <= ed) return ed - st + 1;
            return 0;
        }
    }
    else if(A){
        // Ax + C = 0
        // x = - C / A
        if( C % A ) return 0;
        else if( x1 <= -C / A && -C / A <= x2) return y2 - y1 + 1;
        else return 0;
    }
    else if(B){
        // By + C = 0
        // y = - C / B
        if(C % B) return 0;
        if( y1 <= -C / B && -C / B <= y2) return x2 - x1 + 1;
        return 0;
    }
    else{
        // C = 0
        if(C) return 0;
        return (x2 - x1 + 1) * (y2 - y1 + 1);
    }
}
```

6.10 NEW exEuclid

```

// Minimizes |X| + |Y|
// Breaks tie with X <= Y
LL exEuclid(LL a, LL b, LL &x, LL &y)
{
    if(b == 0){
        x = 1;
        y = 0;
        return a;
    }

    LL nx, ny, g, r;
    g = exEuclid(b, a % b, nx, ny);
    x = ny;
    y = nx - a / b * ny;

    return g;
}

LL __lower_bound(LL x0, LL dx, LL x)
{
    LL d = x - x0;
    if(d > 0 && d % dx)
```

```
    {
        if(dx > 0) return d/dx + 1;
        return d/dx - 1;
    }
    return d/dx;
}
```

6.11 NEW nCr Any Mod

```

namespace NT{

    int ocr(LL n, int p){
        int ret = 0;
        while(n){
            ret += n/p;
            n /= p;
        }
        return ret;
    }

    LL ip(LL a, LL p, int MOD){
        if(!p) return 1 % MOD;
        if(p & 1) return (a * ip(a, p - 1, MOD)) % MOD;
        LL ret = ip(a, p/2, MOD);
        return (ret * ret) % MOD;
    }

    LL F(LL n, int p, int MOD){ //This loops inside this function can be
        optimized with O(MOD * MOD) memory
        if(!n) return 1 % MOD;

        LL c = 1, ret;
        for(int i = 1; i <= min(n, (LL) MOD); i++){
            if(i % p == 0) continue;
            c = (c * i) % MOD;
        }

        LL complete = n / MOD;
        ret = ip(c, complete, MOD);

        for(LL i = complete * MOD + 1; i <= n; i++ ){
            if(i % p == 0) continue;
            ret = (ret * (i % MOD)) % MOD;
        }

        return (ret * F(n/p, p, MOD)) % MOD;
    }


    LL modular_inverse(int a, int p, int n){
        return ip(a, n - n / p - 1, n);
```

```
43      }
44
45      vector<int> p;
46      vector<int> e;
47      vector<int> num;
48      vector<int> rm;
49
50      void init(int MOD)
51      {
52          int i, s = sqrt(MOD);
53          for(i = 2; i <= s; i++)
54          {
55              if(MOD % i == 0){
56                  p.push_back(i);
57                  e.push_back(0);
58                  num.push_back(1);
59                  while(MOD % i == 0){
60                      e.back()++;
61                      num.back() *= i;
62                      MOD /= i;
63                  }
64
65                  s = sqrt(MOD);
66              }
67          }
68          if(MOD != 1){
69              p.push_back(MOD);
70              e.push_back(1);
71              num.push_back(MOD);
72          }
73      }
74
75      LL crt(){
76          LL M = 1, ret = 0, c, b;
77          for(int i = 0; i < (int) num.size(); i++)
78              M = M * num[i];
79
80          for(int i = 0; i < (int) num.size(); i++){
81              b = modular_inverse(M / num[i], p[i], num[i]);
82              c = (b * rm[i]) % M;
83              c = (c * M/num[i]) % M;
84              ret = (ret + c) % M;
85          }
86          return ret;
87      }
88
89
90      int nCr(LL n, LL r, int MOD) // O( (pi ^ ei) * log n ) -> for maximum pi ^
            ei
91      {
92          if(n == r ||r == 0) return 1 % MOD;
```

```
93
94          init(MOD);
95          for(int i = 0; i < (int) p.size() ; i++)
96          {
97              int x = ocr(n, p[i]) - (ocr(r, p[i]) + ocr(n - r, p[i]));
98
99
100             if(x >= e[i]) rm.push_back(0);
101             else{
102                 rm.push_back(1);
103                 while(x--) rm.back() *= p[i];
104
105                 rm.back() = (rm.back() * F(n, p[i], num[i])) % num[i];
106                 rm.back() = (rm.back() * modular_inverse(F(r, p[i], num[i]), p
                        [i], num[i])) % num[i];
107                 rm.back() = (rm.back() * modular_inverse(F(n - r, p[i], num[i
                        ]), p[i], num[i])) % num[i];
108             }
109         }
110
111         return crt();
112     }
113
114     void clear(){
115         p.clear();
116         e.clear();
117         num.clear();
118         rm.clear();
119     }
120 }
```

6.12 NTT

```
1
2  ///Number Theoretic Transformation
3  const int mod = 7340033;
4  const int root = 71;                  /// set it equal to get_root
5  const int root_1 = 413523;            /// set it to ip(root, mod - 2)
6  const int root_pw = 1<<20;            /// mod = c * 2^k + 1 => root_pw = 2^k
7
8  LL ip(LL a, LL p){
9      if(!p) return 1;
10     if(p & 1) return ( a * ip(a, p - 1)) % mod;
11     LL ret = ip(a, p/2);
12     return (ret * ret) % mod;
13 }
14
15 int get_root(){
16     int r, q;
17     for(r = 2; r < mod; r++){
18         if(ip(r, root_pw) == 1){
19             for(q = 1; q < root_pw; q++)
```

```
                if(ip(r, q) == 1) break;

            if(q == root_pw) return r;
        }
    }
    return -1;
}

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

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

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

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

    fft (fa, false),  fft (fb, false);
    for (size_t i=0; i<n; ++i)
```

```
        fa[i] = ((LL) fa[i] * fb[i]) % mod;
    fft (fa, true);

    res.resize (n);
    for (size_t i=0; i<n; ++i)
        res[i] = fa[i];
}
```

6.13 NTT USING FFT

```


///NTT - By Zahin Vai

#define SGT_MAX 65536 /// 2 * MAX at least
typedef complex <double> complx; /// Replace double with long double if more
    precision is required

namespace fft{
    int len, last = -1, step = 0, rev[SGT_MAX];
    long long C[SGT_MAX], D[SGT_MAX], P[SGT_MAX], Q[SGT_MAX], tA[SGT_MAX], tB[
        SGT_MAX];
    complx u[SGT_MAX], v[SGT_MAX], f[SGT_MAX], g[SGT_MAX], dp[SGT_MAX], inv[
        SGT_MAX];

    inline complx conj(complx x){
        return {x.real(), -x.imag()};
    }

    inline long long round_half_even(long double x){
        long long res = abs(x) + 0.5;
        if (x < 0) res = -res;
        return res;
    }

    /// 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) < SGT_MAX; 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;
                }
            }
        }
    }

    /// Modular multiplication
    int mod_multiply(int a, long long* A, int b, long long* B, int mod){
        build(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(u, f, dp);
        for (int i = 0; i < len; i++) g[i] = f[i];
        transform(v, g, dp);

        for (int i = 0; i < len; i++){
            int j = (len - 1) & (len - i);
            complx c1 = conj(f[j]), c2 = conj(g[j]);

            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(u, f, dp);
        transform(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].imag() +
                0.5;
            A[i] = (x + ((y % mod) << 15) + ((z % mod) << 30)) % mod;
        }
        return a + b - 1;
    }

    void multiply(vector<int> &a, vector <int> &b, vector<int> &res, int mod){
        int sA = a.size();
        for(int i = 0; i < sA; i++) tA[i] = a[i];

        int sB = b.size();
        for(int i = 0; i < sB; i++) tB[i] = b[i];
        int degree = mod_multiply(sA, tA, sB, tB, mod);

        res.resize(degree);
        for(int i = 0; i < degree; i++) res[i] = tA[i];
        while(!res.back()) res.pop_back();
    }
}
```

6.14 POLARD RHO

```
/**
    Dependencies:
        1. MillerRabin
    How to Use it?
        1. Call pollardRho.clear();
        2. Call pollardRho.getPrimeFactorization(n);
    See sample main() function below
*/

class PollardRho {
    private:

    MillerRabin millerRabin;

    int prime[50000], status[50000], primeSize;
    void sieve() {
        primeSize = 0;
        memset( status, 0, sizeof status );

        status[0] = status[1] = 1;
        int n = 46340;
```

```
        for ( int i = 4; i <= n; i += 2 ) status[i] = 1;

        int sqrtn = sqrt(n);
        for ( int i = 3; i <= sqrtn; i += 2 ){
            for ( int j = i*i; j <= n; j += 2 * i ) {
                status[j] = 1;
            }
        }

        prime[primeSize++] = 2;
        for ( int i = 3; i <= n; i += 2 ) {
            if ( status[i] == 0 ) {
                prime[primeSize++] = i;
            }
        }
    }

    void factorizeWithSieve(int n) {
        int sqrtn = sqrt(n);
        for ( int i = 0; i < primeSize && prime[i] <= sqrtn; i++ ) {
            if ( n % prime[i] == 0 ) {
                while ( n % prime[i] == 0 ) {
                    factors.push_back(prime[i]);
                    n /= prime[i];
                }
                sqrtn = sqrt(n);
            }
        }
        if ( n != 1 ) {
            factors.push_back(n);
        }
    }

    vlong pollard_rho( vlong n, vlong c ) {
        vlong y = 2, i = 1, k = 2, d;
        __int128 x = 2;
        while (true) {
            x = x * x % n + c;
            if (x >= n) x -= n;
            d = __gcd((vlong)x - y, n);
            if (d > 1) return d;
            if (++i == k) {
                y = x, k <<= 1;
            }
        }
        return n;
    }

    void factorize(vlong n) {
        if (n == 1)
            return ;
```

```
        if (n < 1e+9) {
            factorizeWithSieve(n);
            return ;
        }
        if (millerRabin.isPrime(n)) {
            factors.push_back(n);
            return ;
        }
        vlong d = n;
        for (int i = 2; d == n; i++) {
            d = pollard_rho(n, i);
        }
        factorize(d);
        factorize(n/d);
    }

    public:

    vector<vlong> factors;

    PollardRho() {
        sieve();
    }

    void clear() {
        factors.clear();
    }

    vector<pair<vlong,int>> getPrimeFactorization(vlong n) {
        factorize(n);
        sort(factors.begin(), factors.end());

        vector<pair<vlong,int>> res;
        for( int i = 0; i < factors.size(); i++ ) {
            vlong p = factors[i];
            int cnt = 1;
            while ( i + 1 < factors.size() && factors[i+1] == p) {
                i++;
                cnt++;
            }
            res.push_back({p,cnt});
        }

        return res;
    }
}pollardRho;

/*****************************/

int main() {
    int n = 1e16+8;
```

```
124
125     pollardRho.clear(); /// Don't forget to clear. Important for multi case.
126     prime<pair<vlong,int>> factors = pollardRho.getPrimeFactorization(n);
127     for ( int i = 0; i < factors.size(); i++ ) {
128         int p = factors[i].first;
129         int a = factors[i].second;
130
131         /// p^a is factor of n
132         /// Do your work here
133     }
134 }
```

6.15 Simplex

```
1
2
3  /*
4      -> Simplex Algorithm
5      -> Ax <= b              [ b is inserted at the end of every row of 'Eq'
           matrix ]
6      -> maximize cx          [ c is inserted at the last row of 'Eq' matrix ]
7      -> 0 based indexing
8  */
9
10 #define maxm 10000              // Number of equations
11 #define maxn 100                // Number of variables
12 double INF = 1e100;
13 double eps = 1e-10;
14 double Eq[maxm+5][maxn+5];      // Holds the coefficient of the equations
15 double _R[maxn+5];              // Does nothing, space crated to store the
       optimal values of the variables.
16
17 int counter = 0;
18 void pivot(int m, int n, double a[maxm+5][maxn+5], int B[maxm+5], int N[maxn
       +5], int r, int c)
19 {
20     int i, j;
21     swap(N[c], B[r]);
22     a[r][c]=1/a[r][c];
23     for (j=0; j<=n; j++)if (j!=c) a[r][j]*=a[r][c];
24     for (i=0; i<=m; i++)if (i!=r)
25         {
26             for (j=0; j<=n; j++)if (j!=c)
27                     a[i][j]-=a[i][c]*a[r][j];
28             a[i][c] = -a[i][c]*a[r][c];
29         }
30     counter++;
31 }
32
33 int feasible(int m, int n, double a[maxm+5][maxn+5], int B[maxm+5], int N[maxn
       +5])
34 {
```

```
    int r, c, i;
    double p, v;
    while (1)
    {
        for (p=INF, i=0; i<m; i++) if (a[i][n]<p) p=a[r=i][n];
        if (p>-eps) return 1;
        for (p=0, i=0; i<n; i++) if (a[r][i]<p) p=a[r][c=i];
        if (p>-eps) return 0;
        p = a[r][n]/a[r][c];
        for (i=r+1; i<m; i++) if (a[i][c]>eps)
            {
                v = a[i][n]/a[i][c];
                if (v<p) r=i, p=v;
            }
        pivot(m, n, a, B, N, r, c);
    }
}

// m    -> number of equations
// n    -> number of variables
// a    -> Co-efficient of the equations
// b    -> Optimal value of the variables are stored here
// ret  -> Maximum value of the objective function is stored at this variable
// returns 0 if no solution, -1 if the region is unbounded, 1 if there exists a finite solution.
int simplex(int m, int n, double a[maxm+5][maxn+5], double b[maxn+5], double& ret)
{
    ret = 0;
    int B[maxm+5], N[maxn+5], r, c, i;
    double p, v;
    for (i=0; i<n; i++) N[i]=i;
    for (i=0; i<m; i++) B[i]=n+i;
    if (!feasible(m, n, a, B, N)) return 0;
    while (1)
    {
        for (p=0, i=0; i<n; i++) if (a[m][i]>p)
                p=a[m][c=i];
        if (p<eps)
        {
            for (i=0; i<n; i++) if (N[i]<n)
                    b[N[i]]=0;
            for (i=0; i<m; i++) if (B[i]<n)
                    b[B[i]]=a[i][n];
            ret = -a[m][n];
            return 1;
        }
        for (p=INF, i=0; i<m; i++) if (a[i][c]>eps)
            {
                v = a[i][n]/a[i][c];
                if (v<p) p=v, r=i;
```

```
            }
        if (fabs(p-INF) < eps) return -1;
        pivot(m, n, a, B, N, r, c);
    }
}
```

6.16 Simpson

```

double a, b;
const int N = 1000*1000;
double s = 0;
double h = (b - a) / N;
for (int i=0; i<=N; ++i) {
    double x = a + h * i;
    s += f(x) * ((i==0 || i==N) ? 1 : ((i&1)==0) ? 2 : 4);
}
s *= h / 3;
```

6.17 decompose

```

//decompose
vector<pll> decompose(LL N)
{
    LL i, s = sqrt(N);
    LL cnt;

    vector <pll> prmf;
    for(i = 2; i <= s; i++)
    {
        if(N%i) continue;
        cnt = 0;
        while(!(N%i)) cnt++, N/=i;
        prmf.pb(mp(i,cnt));
        s = sqrt(N);
    }
    if(N != 1) prmf.pb(mp(N,1));

    return prmf;
}
```

6.18 derangement

```

//derangement
int derangement(int n)
{
    if(!n) return n;
    if(n <= 2) return n-1;
    return (n-1)*(derangement(n-1) + derangement(n-2));
}
```

6.19 drng esp

```

//derangement_esp
int derangement(int i, int j)
{
    if(i == 0) return 1;
    if(i == 1) return j-1;
    if(i == 2) return (j-1 + (j-2)*(j-2))%MOD;

    if(dr[i][j] != -1) return dr[i][j];
    return  dr[i][j] = (((LL)(j-i)*derangement(i-1,j-1))%MOD + ((LL)(i-1)*
        derangement(i-2,j-2))%MOD + ((LL)(i-1) * derangement(i-1,j-1))%MOD)%
        MOD;
}
```

6.20 extended euclid

```

//Extended Euclid
pii ex_euclid(int a, int b) // a'x'+b'y' = gcd(a,b) //minimized |x| + |y|
{
    if(a == 0) return mp(0,1);
    pii pr = ex_euclid(b%a, a), ret;
    ret.fi = pr.se - (b / a) * pr.fi;
    return mp(ret.fi, pr.fi);
}
```

6.21 hyperbloc diophantine eqn

```
bool isValidSolution ( int a, int b, int c, int p, int div ) {
    if ( ( ( div - c )% a ) != 0 ) return false; //x = (div - c) / a
    if ( ( (p-b*div) % (a*div) ) != 0 ) return false;// y = (p-b*div) /(a*div)
    return true;
}

//Axy+Bx+Cy=D
int hyperbolicDiophantine ( int a, int b, int c, int d ) {
    int p = a * d + b * c;

    if ( p == 0 ) { //ad + bc = 0
        if ( -c % a == 0 ) return -1; //Infinite solutions (-c/a, k)
        else if ( -b % a == 0 ) return -1; //Infinite solutions (k, -b/a)
        else return 0; //No solution
    }
    else {
        int res = 0;

        //For each divisor of p
        int sqrtn = sqrt ( p ), div;
        for ( int i = 1; i <= sqrtn; i++ ) {
            if ( p % i == 0 ) { //i is a divisor
```

```
23
24                  //Check if divisors i,-i,p/i,-p/i produces valid solutions
25                  if ( isValidSolution(a,b,c,p,i) )res++;
26                  if ( isValidSolution(a,b,c,p,-i) )res++;
27                  if ( p / i != i ) { //Check whether p/i is different divisor
                        than i
28                      if ( isValidSolution(a,b,c,p,p/i) )res++;
29                      if ( isValidSolution(a,b,c,p,-p/i) )res++;
30                  }
31              }
32          }
33
34          return res;
35      }
36  }
```

6.22 number of solutions ex euclid

```
1
2  LL go(LL a, LL b, LL p) // 0 < min(a,b,p) && max(a,b,p) < 1e8
3  {
4      int i, j;
5      pair<LL, LL> init;
6      LL l, r, n, g = __gcd(a,b), m = p/g;
7      if(p%g) return 0;
8      init = ex_euclid(a,b);
9      init.xx *= m;
10     init.yy *= m;
11     if(init.xx > 0)
12     {
13         n = g*init.xx;
14         n = n/b+1;
15     }
16     else n = 1;
17     init.xx -= (b/g) * n;
18     init.yy += (a/g) * n;
19     l = (-init.xx * g);
20     if(l % b) l = l/b + 1;
21     else l = l/b;
22     r = init.yy * g;
23     r = r/a + 1;
24     return max(0LL, r-l);
25 }
```

6.23 phi big

```
1
2  //phi
3  LL phi(LL N)
4  {
5      vector<pll> nw;
6      LL ret = N;
```

```
    nw = decompose(N);

    LL i;
    for(i = 0; i < nw.size(); i++)
    {
        ret /= nw[i].fi;
        ret *= (nw[i].fi - 1);
    }

    return ret;
}
```

6.24 relative prime counter

```

//Number of Relative primes
LL phi[MAX+10]; // must be LL
void calc_phi(int n)
{
    int i, j;
    phi[1] = 1; // may be 0, if needed
    for(i = 2; i <= n; i++) phi[i] = i;
    for(i = 2; i <= n; i++)
        if(phi[i] == i)
            for(j = i; j <= n; j += i) phi[j] *= (i-1), phi[j]/=i;
}
```

6.25 roots of a polynomial

```

// Computes the real roots of a n-degree polynomial
#define eps             1e-9
#define DEPTH_LIMIT     100     // Terminates the process in case of existence
    of imaginary roots

long double random(long double low, long double high, int itr = 10){
    /*
        returns a double value between the range low and high
        decrease the parameter itr if you just got TLE
    */
    uniform_real_distribution<long double> unif(low, high);
    default_random_engine re;

    int x = rand() % itr;
    while(x--) unif(re);
    return unif(re);
}

void normalize(vector<long double> &polynomial){
    /*
        removes the leading zeros
    */
```

```
    while(polynomial.size() && fabs(polynomial.back()) < eps) polynomial.
        pop_back();
}


long double eval(vector<long double> polynomial, long double x){
    long double ret = 0, p = 1;
    for(auto c : polynomial)
    {
        ret += c * p;
        p = p * x;
    }
    return ret;
}

vector<long double> divide(vector<long double> polynomial, long double z){
    /*
        divides the polynomial by (x - z)
        (x-z) must divide the polynomial
    */
    vector<long double> ret;
    int i;
    for(i = (int) polynomial.size() - 1; i > 0; i--){
        if(i == (int) polynomial.size() - 1) ret.push_back(polynomial[i]);
        else ret.push_back(polynomial[i] + ret.back() * z);
    }

    reverse(ret.begin(), ret.end());
    normalize(ret);
    return ret;
}

vector<long double> differentiate(vector<long double> polynomial){
    vector<long double> ret;
    int i;
    for(i = 1; i < (int) polynomial.size(); i++)
        ret.push_back(polynomial[i] * i);

    return ret;
}


vector<long double> nothing; // Just an empty vector
vector<long double> newton_raphson(vector<long double> polynomial, vector<long
    double> derivative, long double low, long double high, int depth, int itr
    = 50){
    /*
        decrease the parameter itr if you just got TLE
        increase the parameter itr if you just got WA
    */

```

```
71      if(depth > DEPTH_LIMIT) return nothing;
72
73      normalize(polynomial);
74      normalize(derivative);
75
76      if(polynomial.size() <= 1) return nothing;
77
78      long double x0 = random(low, high);
79
80      while(itr--){
81          long double up = eval(polynomial, x0);
82          long double dwn = eval(derivative, x0);
83
84          if(fabs(dwn) < eps) {return newton_raphson(polynomial, derivative, low
                , high, depth + 1);}
85          x0 = x0 - up/dwn;
86      }
87
88
89      if(abs(eval(polynomial, x0)) < eps){
90          polynomial = divide(polynomial, x0);
91          derivative = differentiate(polynomial);
92
93          vector<long double> ret = newton_raphson(polynomial, derivative, low,
                high, depth + 1);
94          ret.push_back(x0);
95          return ret;
96      }
97      return newton_raphson(polynomial, derivative, low, high, depth + 1);
98  }
99
100 vector<long double> solve(vector<long double> polynomial, long double low,
        long double high){
101     /*
102         this is the function to be called from main
103         polynomial -> f(x) = p[0] + p[1] * x + p[2] * x^2 + ..... p[n-1] * x
                ^(n - 1)
104         [low, high] -> the range where the roots can exist.
105     */
106     if(polynomial.size() <= 1) return nothing;
107
108     vector<long double> ret, tmp;
109     vector<long double> derivative = differentiate(polynomial);
110     tmp = solve(derivative, low, high);
111
112     for(auto x : tmp){
113         if(fabs(eval(polynomial, x)) < eps){
114             ret.push_back(x);
115             polynomial = divide(polynomial, x);
116         }
117     }
```

```
    tmp = newton_raphson(polynomial, differentiate(polynomial), low, high, 0);
    for(auto x : tmp)
        ret.push_back(x);

    return ret;
}
```

6.26 shanks

```

//Shank's Baby Step Gaint Step
int shanks(int a, int b, int p) // a^x = b (mod m), x = ?
{
    int i, rp;
    LL baby, ai = 1, j;
    rp = (int) ceil(sqrt(p));

    map<LL, int>::iterator it;
    map<LL, int> M;
    M.insert(mp(1,0));

    for(i = 1; i < rp; i++)
    {
        ai = ((LL)ai*a)%p;
        M.insert(mp(ai,i));
    }

    baby = ip(ip(a,p-2,p), rp, p); // a ^ (-m)
    for(j = b, i = 0; i < rp; i++)
    {
        it = M.find(j);
        if(it != M.end()) return i*rp + it->second;
        j = (j*baby)%p;
    }
    return 0;
}
```

6.27 sum of divisors upto n

```

//Sum of Divisors upto N

LL sum(LL L, LL R) {return (R * (R+1))/2 - (L * (L-1))/2;}

LL sodUpto(LL n)
{
    int s = sqrt(n), i;
    LL x_min, x_max, ret = 0;

    for(i = 1; i <= s; i++)
        ret += i * (n/i);
```

```
    if(s * s == n) s--;
    for(i = 1; (LL) i <= s; i++)
    {
        x_min = max((LL)s + 1, n/(i+1) + 1);
        x_max = n/i;

        if(x_min > x_max) continue;
        ret += sum(x_min, x_max) * i;
    }
    return ret;
}
```

7 String

7.1 kmp

```
//KMP
char ptrn[MAX+10], text[MAX+10];
int F[MAX+10];

void failure_function(char *pat)
{
    int len = strlen(pat), idx, i;

    F[0] = F[1] = 0; // F[i] = length of the longest PROPER suffix ending at
        pat[i-1] which is also a PROPER prefix
    for(idx = 2; idx <= len; idx++)
    {
        i = F[idx-1];
        while(1)
        {
            if(pat[i] == pat[idx-1]){ F[idx] = i+1; break;}
            else if(i) i = F[i];
            else {F[idx] = 0; break;}
        }
    }
}

int kmp(char *txt, char *pat) // both 0 indexed
{
    failure_function(pat);
    int txt_len = strlen(txt), pat_len = strlen(pat), pid, ti, ret = 0;
    if(pat_len > txt_len) return 0;

    for(ti = pid = 0; ti < txt_len; ti++)
    {
        while(1)
        {
            if(txt[ti] == pat[pid])
            {
```

```
                pid++;
                if(pid == pat_len) ret++, pid = F[pid]; // Match Found at
                    position ti
                break;
            }
            else if(pid) pid = F[pid];
            else break;
        }
    }
    return ret;
}
```

7.2 manachar

```
//manacher

vector<int> man_odd(char *s)
{
    int n = strlen(s);
    vector<int> d1 (n);
    int l=0, r=-1;
    for (int i=0; i<n; ++i) {
        int k = (i>r ? 0 : min (d1[l+r-i], r-i)) + 1;
        while (i+k < n && i-k >= 0 && s[i+k] == s[i-k])  ++k;
        d1[i] = k--;
        if (i+k > r)
            l = i-k,  r = i+k;
    }

    return d1;
}

vector<int> man_even(char *s)
{
    int n = strlen(s);
    vector<int> d2 (n);
    int l=0, r=-1;
    for (int i=0; i<n; ++i) {
        int k = (i>r ? 0 : min (d2[l+r-i+1], r-i+1)) + 1;
        while (i+k-1 < n && i-k >= 0 && s[i+k-1] == s[i-k])  ++k;
        d2[i] = --k;
        if (i+k-1 > r)
            l = i-k,  r = i+k-1;
    }

    return d2;
}
```

7.3 suffix array

```

// Suffix_Array O( n * lgn * lgn)
```

```
// 0 based indexing

int Plc[MAXLG+5][MAXN+10], stp;
int S[MAXN+10]; //Sorted Suffixes
pair< pii , int> L[MAXN+10];

void Generate_SA(string str)
{
    int i, j, k, len = str.size(), cur;
    for(i = 0; i < len; i++)    Plc[0][i] = str[i];

    for(cur = stp = 1; (cur>>1) < len; cur *= 2, stp++)
    {
        for(i = 0; i < len; i++)
        {
            L[i].xx.xx = Plc[stp-1][i];
            L[i].xx.yy = i+cur < len? Plc[stp-1][i+cur]:-inf; // set it to -
                inf when dealing with negative numbers
            L[i].yy = i;
        }
        sort(L, L+len);
        for(i = 0; i < len; i++)
        {
            if( !i || L[i-1].xx.xx != L[i].xx.xx || L[i-1].xx.yy != L[i].xx.yy
                ) Plc[stp][L[i].yy] = i;
            else Plc[stp][L[i].yy] = Plc[stp][L[i-1].yy];
        }
    }
    for(i = 0; i < len; i++)
        S[Plc[stp-1][i]] = i;
    stp--;
}

int lcp(int u, int v, int N) // Here N = length of the string **Call
    Generate_SA(string)**
{
    int ret = 0, k;
    if(u == v) return N-u;
    for(k = stp; k >= 0 && u < N && v < N; k--)
        if(Plc[k][u] == Plc[k][v])
            ret += 1<<k, u += 1<<k, v += 1<<k;
    return ret;
}
```

7.4 trie dynamic

```

///Memory_Efficient
// Dynamic Trie
// Slower than static Trie
// have a root = new trie

```

```
7  #define AS          26
8  #define scale(x)    x-'a'
9
10 struct trie{
11     trie *nxt[AS+2];
12     int ep;
13
14     trie()
15     {
16         int i;
17         ep = 0;
18         for(i = 0; i <= AS; i++)
19             nxt[i] = NULL;
20     }
21 };
22
23 void Insert(trie *rt, char *s)
24 {
25     int i, v, len = strlen(s);
26     for(i = 0; i < len; i++)
27     {
28         v = scale(s[i]);
29         if(rt->nxt[v] == NULL)
30             rt->nxt[v] = new trie;
31
32         rt = rt->nxt[v];
33     }
34     rt->ep++;
35 }
36
37 bool Find(trie *rt, char *s)
38 {
39     int i, v, len = strlen(s);
40     for(i = 0; i < len; i++)
41     {
42         v = scale(s[i]);
43         if(rt->nxt[v] == NULL) return false;
44         rt = rt->nxt[v];
45     }
46     return rt->ep;
47 }
48
49 void rmv(trie *rt)
50 {
51     int i;
52     for(i = 0; i <= AS; i++)
53         if(rt->nxt[i] != NULL)
54             rmv(rt->nxt[i]);
55     free(rt);
56 }
```

7.5 trie static

```
1
2  ///Time_Efficient
3  // Static TrieTree
4  // AS => Alphabet Size
5
6  #define MAXNODE     1000000
7  #define AS          26
8  #define scale(x)    (x-'a')
9
10 int nxt[MAXNODE+2][AS+2];
11 int ep[MAXNODE+2];
12
13 struct TrieTree{
14     int idx, rt;
15     TrieTree()
16     {
17         rt = idx = 0;
18         memset(nxt[rt], -1, sizeof(nxt[rt]));
19         memset(ep, false, sizeof(ep));
20     }
21     void insert(char *s);
22     bool find(char *s);
23 };
24
25 void TrieTree::insert(char *s)
26 {
27     int i, cur, len = strlen(s), v;
28
29     cur = rt;
30     FRL(i,0,len)
31     {
32         v = scale(s[i]);
33         if(nxt[cur][v] == -1)
34         {
35             idx++;
36             memset(nxt[idx], -1, sizeof(nxt[idx]));
37             nxt[cur][v] = idx;
38         }
39         cur = nxt[cur][v];
40     }
41     ep[cur]++;
42 }
43
44 bool TrieTree::find(char *s)
45 {
46     int i, cur, len = strlen(s), v;
47
48     cur = rt;
49     FRL(i,0,len)
```

```
    {
        v = scale(s[i]);
        if(nxt[cur][v] == -1) return false;
        cur = nxt[cur][v];
    }
    return ep[cur];
}
```

7.6 z function

```

// Z-function
#define MAX 100000
char str[MAX+10];
int z[MAX+10];

void z_function() // z[i] = length of the longest substring starting from i
    that is also a prefix of str
{
    int n = strlen(str);
    z[0] = n;
    for (int i=1, l=0, r=0; i<n; ++i)
    {
        z[i] = 0;
        if (i <= r)
            z[i] = min (r-i+1, z[i-l]);
        while (i+z[i] < n && str[z[i]] == str[i+z[i]])
            ++z[i];
        if (i+z[i]-1 > r)
            l = i,  r = i+z[i]-1;
    }
}
```

8 z Others

8.1 example lazy

```

/*
Given a_1, a_2, a_3, a_4..... a_n
Supports two operation:
1. Add L R X: adds X to a_L, a_(L+1), ..... a_R
2. Query L R: returns fib(a_L) + fib(a_(L+1)) + ..... + fib(a_R)
*/

#include<bits/stdc++.h>
using namespace std;
#define MOD     1000000007
#define MAX     100000
#define D(x)    cout << #x " = " << (x) << endl
typedef long long int LL;

```

```
16 inline LL mul(LL x, LL y){
17     LL ret = x * y;
18     if(ret >= MOD) ret %= MOD;
19     return ret;
20 }
21
22 inline LL add(LL u, LL v){
23     LL ret = u + v;
24     if(abs(ret) >= MOD) ret %= MOD;
25     if(ret < 0) ret += MOD;
26
27     return ret;
28 }
29
30
31 struct matrix{
32     int mat[2][2], dim;
33     matrix(){}
34
35     matrix operator * (const matrix &r){
36         int i, j, k;
37         matrix ret;
38
39         for(i = 0; i < 2; i++)
40             for(j = 0; j < 2; j++)
41             {
42                 ret.mat[i][j] = 0;
43                 for(k = 0; k < 2; k++)
44                     ret.mat[i][j] = add(ret.mat[i][j], mul(mat[i][k], r.mat[k
                        ][j]));
45             }
46         return ret;
47     }
48
49 };
50
51 matrix unitMatrix(){
52     matrix ret;
53     ret.mat[0][0] = ret.mat[1][1] = 1;
54     ret.mat[0][1] = ret.mat[1][0] = 0;
55     return ret;
56 }
57
58 matrix expo(matrix &in, LL p)
59 {
60     matrix ret = unitMatrix(), aux = in;
61     while(p)
62     {
63         if(p&1) ret = ret*aux;
64         aux = aux*aux;
65         p>>=1;
```

```
66      }
67      return ret;
68  }
69
70  matrix fibMatrix(int p){
71      matrix ret;
72      ret.mat[0][0] = ret.mat[0][1] = ret.mat[1][0] = 1;
73      ret.mat[1][1] = 0;
74
75      return expo(ret, p);
76  }
77
78  bool notCleared(matrix m){
79      if(m.mat[0][0] != 1) return true;
80      if(m.mat[0][1] != 0) return true;
81      if(m.mat[1][0] != 0) return true;
82      if(m.mat[1][1] != 1) return true;
83      return false;
84  }
85
86  int arr[MAX+5];
87
88  struct lazy{
89      matrix M;
90
91      lazy(){
92          M = unitMatrix();
93      }
94      void compose(lazy ano){
95          M = M * ano.M;
96      }
97  };
98
99  struct node{
100     int sW, sX;
101     lazy L;
102
103     node(){
104         sW = sX = 0;
105     }
106
107     void apply(lazy _L){
108         int t_sW = sW;
109         int t_sX = sX;
110
111         sW = mul( sW , _L.M.mat[0][0]);
112         sW = add(sW , mul(sX, _L.M.mat[1][0]));
113
114         sX = mul(t_sW, _L.M.mat[0][1]);
115         sX = add(sX, mul( t_sX, _L.M.mat[1][1]));
116
```

```
117         L.compose(_L);
118     }
119 
120     void clearLazy(){
121         L.M = unitMatrix();
122     }
123 } tree[MAX << 2];
124 
125 
126 void propagate(int idx, int st, int ed)
127 {
128     int mid = (st+ed)/2, l = 2*idx, r = l + 1;
129 
130     tree[l].apply(tree[idx].L);
131     tree[r].apply(tree[idx].L);
132 
133     tree[idx].clearLazy();
134 }
135 
136 void build_tree(int idx, int st, int ed)
137 {
138     if(st == ed){
139         tree[idx].L.M = fibMatrix(arr[st]);
140         tree[idx].sW = tree[idx].L.M.mat[1][0];
141         tree[idx].sX = tree[idx].L.M.mat[1][1];
142 
143         return;
144     }
145     int mid = (st+ed)/2, l = 2*idx, r = l + 1;
146     build_tree(l, st, mid);
147     build_tree(r, mid+1, ed);
148 
149     tree[idx].sX = add(tree[l].sX, tree[r].sX);
150     tree[idx].sW = add(tree[l].sW, tree[r].sW);
151 }
152 
153 void update(int idx, int st, int ed, int i, int j, lazy &curr)
154 {
155     if(st == i && ed == j)
156     {
157         tree[idx].apply(curr);
158         return;
159     }
160 
161     int mid = (st+ed)/2, l = 2*idx, r = l+1;
162     if( notCleared(tree[idx].L.M) ) propagate(idx, st, ed);
163 
164     if(j <= mid) update(l , st, mid, i, j, curr);
165     else if(i > mid) update(r, mid+1, ed, i, j, curr);
166     else update(l, st, mid, i, mid, curr), update(r, mid+1, ed, mid+1, j, curr
        );
```

```
167
168     tree[idx].sX = add(tree[l].sX, tree[r].sX);
169     tree[idx].sW = add(tree[l].sW, tree[r].sW);
170 }
171
172 int query(int idx, int st, int ed, int i, int j)
173 {
174     if(st == i && ed == j) return tree[idx].sW;
175     int mid = (st+ed)/2, l = 2*idx, r = l + 1;
176     if( notCleared(tree[idx].L.M) ) propagate(idx, st, ed);
177
178     if(j <= mid) return query(l, st, mid, i, j);
179     if(i > mid) return query(r, mid+1, ed, i, j);
180     return (query(l,st,mid,i,mid) + query(r,mid+1,ed,mid+1,j))%MOD;
181 }
182
183 int main()
184 {
185     //freopen("in.txt", "r", stdin);
186
187     int i, j, k, n, q;
188     int l, r, x, tp;
189     lazy L;
190
191     scanf("%d %d", &n, &q);
192     for(i = 1; i <= n; i++)
193         scanf("%d", arr + i);
194
195     build_tree(1, 1, n);
196
197     while(q--)
198     {
199         scanf("%d", &tp);
200         if(tp == 1){
201             scanf("%d %d %d", &l, &r, &x);
202             L.M = fibMatrix(x);
203             update(1, 1, n, l, r, L);
204
205         }
206         else{
207             scanf("%d %d", &l, &r);
208             printf("%d\n", query(1, 1, n, l, r));
209         }
210     }
211     return 0;
212 }
```

8.2 fast io any os

```
1 //FAST IO
2 //Remember to fix data type
3 //Doesn't work for negative numbers
```

```
int readint()
{
   int cc = getc(stdin);
   for (;cc < '0' || cc > '9';)  cc = getc(stdin);
   int ret = 0;
   for (;cc >= '0' && cc <= '9';)
   {
      ret = ret * 10 + cc - '0';
      cc = getc(stdin);
   }
  return ret;
}
```

8.3 fast io linux

```

inline void fastRead_int(int &x)
{
    register int c = getchar_unlocked();
    x = 0;
    int neg = 0;
    for(; ((c<48 || c>57) && c != '-'); c = getchar_unlocked());

    if(c=='-'){
        neg = 1;
        c = getchar_unlocked();
    }

    for(; c>47 && c<58 ; c = getchar_unlocked()){
        x = (x<<1) + (x<<3) + c - 48;
    }

    if(neg)
        x = -x;
}


inline void fastRead_string(char *str)
{

    register char c = 0;
    register int i = 0;

    while (c < 33)
        c = getchar_unlocked();

    while (c != '\n')
    {
        str[i] = c;
        c = getchar_unlocked();
        i = i + 1;
```

```
    }
    str[i] = '\0';
}

inline void print(int a)
{
    char s[11];
    int t = -1;
    do
    {
        s[++t] = a % 10 + '0';
        a /= 10;
    }
    while(a > 0);
    while(t >= 0)putchar_unlocked(s[t--]);
    putchar_unlocked('\n');
}
```

8.4 fibonacchi large

```

///Fibonacci_large
#define MOD 1000000007
map<LL, LL> Fib;
void init(){Fib[0] = Fib[1] = 1;}
LL f(LL n)  //will return (n-1)th Fibonacci number
{
    if(n == -1) return 0;
    if (Fib.count(n)) return Fib[n];
    long k=n/2;
    if (!(n&1)) return Fib[n] = (f(k)*f(k) + f(k-1)*f(k-1)) % MOD;
    else return Fib[n] = (f(k)*f(k+1) + f(k-1)*f(k)) % MOD;
}
```