AlCocoRekt

November 1, 2016

Contents

1 DP and Math

1.1 DP Optimizations

1.1.1 ConvexHullTrick-hashed

```
f425 #include "bits/stdc++.h"
3d1e using namespace std;

43e7 const int N = 3e5 + 5;


e461 /** ----- Minimum Convex Hull Trick Template ------- */

fd7b struct cht{
2ee8   struct line{
4fa4          long long a , b;
571a          double xleft;
79ae          bool type;
c994          line(long long _a , long long _b){
c5e0                 a = _a;
d7e6                 b = _b;
681b                 type = 0;
db3d          }
087a          bool operator < (const line &other) const{
aac5                 if(other.type){
1ff8                        return xleft < other.xleft;
c082                 }
823d                 return a > other.a;
ec6c          }
f303   };
ecb0   inline double intersect(line x , line y){
e964          return 1.0 * (y.b - x.b) / (x.a - y.a);
2736   }
7689   multiset < line > hull;
2931   cht(){
5487          hull.clear();
3ad9   }
d0ac   typedef set < line > :: iterator iter;
3efe   inline bool has_left(iter node){
342c          return node != hull.begin();
61dc   }
bb43   inline bool has_right(iter node){
a4c6          return node != prev(hull.end());
355b   }
ffe4   inline void update_border(iter node){
0079          if(has_right(node)){
cdc3                 line temp = *next(node);
64cd                 hull.erase(hull.find(temp));
369e                 temp.xleft = intersect(*node, temp);
8f81                 hull.insert(temp);
0c01          }
e543          if(has_left(node)){
fe86                 line temp = *node;
b136                 temp.xleft = intersect(*prev(node), temp);
059a                 hull.erase(node);
4e8d                 hull.insert(temp);
6a09          }
60d3          else{
4c96                 line temp = *node;
0277                 hull.erase(node);
dde4                 temp.xleft = -1e18;
513b                 hull.insert(temp);
daf4          }
a6aa   }
f56a   inline bool useless(line left , line middle , line right){
cfc0          return intersect(left, middle) > intersect(middle, right);
0603   }
76ab   inline bool useless(iter node){
4596          if(has_left(node) && has_right(node)){
4462                 return useless(*prev(node), *node, *next(node));
125e          }
eed1          return 0;
8f0b   }
       // add line with equation y = (a * x + b)
b262   inline void add(long long a , long long b){
32f9          line temp = line(a, b);
443c          auto it = hull.lower_bound(temp);
84bd          if(it != hull.end() && it -> a == a){
9ebb                 if(it -> b > b){
```

```
1e01              hull.erase(it);
088d            }
22e2            else{
5b41              return;
0aa7            }
3828        }
2202        hull.insert(temp);
2e96        it = hull.find(temp);
db9c        if(useless(it)){
bae9            hull.erase(it);
2840            return;
013f        }
350b        while(has_left(it) && useless(prev(it))){
e130            hull.erase(prev(it));
8774        }
69ba        while(has_right(it) && useless(next(it))){
0e65            hull.erase(next(it));
280e        }
0700        update_border(it);
0045    }
        // get minimum value of (m * x + c) for given x
6e3f    inline long long query(long long x){
a577        if(hull.empty()){
120b            return 5e18;
58ed        }
d641        line query(0, 0);
7b4c        query.xleft = x;
90dd        query.type = 1;
aa9e        auto it = hull.lower_bound(query);
a56f        it = prev(it);
ea70        return it -> a * x + it -> b;
872e    }
2399  };


9baf  /** ----------- End of Template ------------ **/


2efc  cht tree[N * 4];

9bef  inline void update_tree(int node, int l, int r, int qs, int qe, int
   m, int c){
3b7a    if(l > qe || r < qs) return;
3f37    if(l >= qs && r <= qe){
e295        tree[node].add(-m, -c);
```

```
a34f        return;
7d67    }
9b38    int mid = (l + r) >> 1;
5fa5    update_tree(node * 2, l, mid, qs, qe, m, c);
f55c    update_tree(node * 2 + 1, mid + 1, r, qs, qe, m, c);
e7d7  }

a1f4  inline long long query_tree(int node, int l, int r, int pos, int x){
629e    long long cur = -tree[node].query(x);
9438    if(l != r && pos){
7267        int mid = (l + r) >> 1;
b2d3        if(mid >= pos) cur = max(cur, query_tree(node * 2, l, mid,
    pos, x));
432e        else cur = max(cur, query_tree(node * 2 + 1, mid + 1, r,
    pos, x));
7264    }
af6b    return cur;
5d06  }

085d  int t, q, x, id;
4006  pair < int, int > query[N], range[N];

0ed2  int main(){
cb98    scanf("%d", &q);
5d9f    for(int i = 1; i <= q; i++){
a372        scanf("%d", &t);
1c7a        if(t == 1){
70d1            scanf("%d %d", &query[i].first, &query[i].second);
4867            range[i] = {i, q};
3a3e        }
0754        else if(t == 2){
4c91            scanf("%d", &id);
37bd            range[id] = {range[id].first, i - 1};
e9c0        }
8412        else{
91c9            scanf("%d", &x);
348a            range[i] = {x, q + 1};
51d9        }
caf3    }
e81f    for(int i = 1; i <= q; i++){
fad2        if(range[i].first == 0 && range[i].second == 0) continue;
dbbb        if(range[i].second <= q){
6b34            update_tree(1, 1, q, range[i].first,
    range[i].second, query[i].first, query[i].second);
a324        }
```

```
f637           else{
052c                   long long res = query_tree(1, 1, q, i,
    range[i].first);
870c                   if(res == -5e18) puts("EMPTY SET");
0ff9                   else printf("%lld\n", res);
c802           }
163d    }
e8cc  }
```

1.1.2 DivideAndConquer-hashed

```
      // Codeforces VK Cup 2016 - Divide and Conquer DP Optimization
      // This can be used when best[i][j] <= best[i + 1][j], where
          best[i][j] is the index (k) which maximises dp[i][j]
      // You can exploit the monotonicity of the best[][] array to speed
          up your code.
8c39  #include <bits/stdc++.h>
3a00  using namespace std;

31d6  const int N = 200005;
39e9  const int K = 55;

131a  int n, k, t[N];
76dd  double t_sum[N], t_inv[N], pre[N], dp[K][N];

782b  /*
2cd8  dp[j][i]  = min(dp[j - 1][k] + cost[k + 1][i]), where (1 <= k < i).
b1e1  cost[i][j] = (t[i]) / (t[i]) +
5680              (t[i] + t[i + 1]) / (t[i + 1]) +
489b              (t[i] + t[i + 1] + t[i + 2]) / (t[i + 2]) + ....
a778              (t[i] + t[i + 1].... + t[j]) / (t[j])
684b            = (pre[j] - pre[i - 1]) - t_sum[i - 1] * (t_inv[j] -
    t_inv[i - 1])
42f4  */

b5b3  inline double cost(int i, int j){
b56e    return (pre[j] - pre[i - 1]) - t_sum[i - 1] * (t_inv[j] - t_inv[i
    - 1]);
75d6  }

a7a8  inline void compute(int j, int l, int r, int qs, int qe){
9f89    int i = (l + r) >> 1, best_idx = -1;
adf4    dp[j][i] = 1e18;
7363    for(int k = qs; k <= min(qe, i - 1); k++){
3f80            if(dp[j - 1][k] + cost(k + 1, i) < dp[j][i]){
f9fc                    dp[j][i] = dp[j - 1][k] + cost(k + 1, i);
abbf                    best_idx = k;
fd20            }
0794    }
4086    if(i > l) compute(j, l, i - 1, qs, best_idx);
fab4    if(i < r) compute(j, i + 1, r, best_idx, qe);
a7a8  }

0ed2  int main(){
1227    scanf("%d %d", &n, &k);
820f    for(int i = 1; i <= n; i++){
ea0b            scanf("%d", t + i);
9a5b            t_sum[i] = t_sum[i - 1] + t[i];
7b85            t_inv[i] = t_inv[i - 1] + (1.0 / t[i]);
2511            pre[i]  = pre[i - 1] + (t_sum[i] / t[i]);
8955    }
697e    for(int i = 1; i <= n; i++) dp[1][i] = dp[1][i - 1] + (pre[i] -
    pre[i - 1]);
7fa9    for(int i = 2; i <= k; i++) compute(i, 1, n, 1, n);
3cf8    printf("%.10f\n", dp[k][n]);
c19a  }
```

1.2 FFT

1.2.1 FFT-hashed

```
f425  #include "bits/stdc++.h"
3d1e  using namespace std;

      // Start of Integer FFT Template.
      // Values are computed modulo 1.6e8
0722  inline long long gcd(long long a, long long b, long long &s, long
    long &t) {
35bf      if (b == 0) {
7cc9          t = 0;
b5ae          s = (a < 0) ? -1 : 1;
043c          return (a < 0) ? -a : a;
4a16      } else {
033a          long long g = gcd(b, a % b, t, s);
9fdf          t -= a / b * s;
d75e          return g;
f6c7      }
3fcb  }

447b  inline long long inverse(long long n, long long mod) {
56b6      long long s, t;
e239      gcd(n, mod, s, t);
0696      return (s > 0) ? s : s + mod;
b049  }

a12d  const long long mod = 5 * (1 << 25) + 1;
7b9e  const long long root = 243;
d5aa  const long long root_1 = 114609789;
8135  const long long root_pw = 1 << 25;

1093  inline void fft (vector < long long > & a, bool invert) {
1fbe      int n = (int) a.size();
e482      for (int i = 1, j = 0; i < n; i++) {
2308          int bit = n >> 1;
e056          for (; j >= bit; bit >>= 1) {
a19d              j -= bit;
ed71          }
4339          j += bit;
ce52          if (i < j) {
9f53            swap(a[i], a[j]);
9c87          }
3c99      }
a04f      for (int len = 2; len <= n; len <<= 1) {
c073          long long wlen = invert ? root_1 : root;
7cac          for (long long i = len; i < root_pw; i <<= 1)
5374              wlen = (long long) (wlen * 1ll * wlen % mod);
4bf6          for (int i = 0; i < n; i += len) {
822c              long long w = 1;
a482              for (int j = 0; j < len / 2; j++) {
e093                  long long u = a[i + j];
6f96                  long long v = (long long) (a[i + j + len / 2] * 1ll *
     w % mod);
f37f                  a[i + j] = u + v < mod ? u + v : u + v - mod;
6c40                  a[i + j + len / 2] = u - v >= 0 ? u - v : u - v + mod;
54c4                  w = (long long) (w * 1ll * wlen % mod);
22db              }
d96b          }
5eb6      }
18ba      if (invert) {
1b79          long long nrev = inverse(n, mod);
ac3e          for (int i = 0; i < n; i++)
ecf9            a[i] = (long long) (a[i] * 1ll * nrev % mod);
cf1a      }
d605  }

      // End to Integer FFT Template.

88f5  const int N = 1000000;
e983  char str[N];

0ed2  int main() {
f0aa      scanf("%s", str);
a2c1      int length = strlen(str);
970d      int size = 1;
b73d      while (size < 2 * length) {
ae35          size *= 2;
ad0c      }
4e42      vector < long long > A(size, 0);
f689      vector < long long > B(size, 0);
3937      for (int i = 0; i < length; i++) {
27b3          if (str[i] == 'A') {
5fb4              A[i] = 1;
2437          } else {
a580              B[length - i] = 1;
0551          }
8857      }
6e91      fft(A, false);
1ef2      fft(B, false);
e2ad      vector < long long > C(size, 0);
0cc3      for (int i = 0; i < size; i++) {
5e55          C[i] = A[i] * B[i] % mod;
aa8f      }
e46c      fft(C, true);
db1e      for (int i = length + 1; i < 2 * length; i++) {
fef9          printf("%lld\n", C[i]);
cf8a      }
5601  }
```

1.3 Gaussian Elimination

1.3.1 GaussianElimination-hashed

```
f425  #include "bits/stdc++.h"
3d1e  using namespace std;
```

```
782b /*
0f28  dp[i] = Expected number of steps to reach the goal (99) from the
  i'th cell.
5d3d  dp[99] = 0.0;

89bc  to[i] -> the cell one reaches if one lands on cell i.

c313  dp[i] = (1 / 6) * (dp[to[i + 1]] + dp[to[i + 2]] ... + dp[to[i +
  6]]) + 1
35a9  Notice that this relation can be cyclic because of the snakes and
  ladders.
52f2  Hence you cannot do simple bottom up dp!

e68e  We can set up n linear equations of the form :-
433a  dp[i] - p1(dp[to[i + 1]]) - p2(dp[to[i + 2]]) ... - p6(dp[to[i +
  6]]) = 1

f522  Coefficients are 1, -p1, -p2...-p6 ------> 1
905a  So you have a system of 100 equations with 100 variables each
  (albeit most
c661  of the coefficients are zero). You have to solve this set of
  equations to
28db  compute the value of dp[0]

2499  It is guaranteed that a solution will exist since the game will
  end in some
528c  finite number of steps.

a62f  Implementation notes :-
53e3  1) to[i + x] = i if (i + x) > 99 (since you stay at the same place)
0244  2) Handle base case separately -> result[99] = 0 since dp[99] = 0
  (special case!)
c12e */

4afb const int N = 100;

99d4 int to[N];
9a73 double coefficients[N][N], result[N];
1c30 int t, n, u, v;

cd27 struct gaussian_elimination{

1746  static const double EPS = 0.000001;
```

```
a4a7   inline double abs_val(double &d){
1baf     return (d < 0.0F) ? -d : d;
78a0   }

1315   inline bool is_zero(double d){
9945     return abs_val(d) < EPS;
2cb7   }

e1c9   static const int VAR = N;

fde2   int n;
0d96   double co[VAR][VAR]; // coefficients
73eb   double ans[VAR]; // value of each variable
cf45   double res[VAR]; // constant part of each equation
55aa   bool rekt; // is true when unique solution does not exist
88ac   bool u[VAR]; // equation already used to eliminate some variable
fe51   int used[VAR]; // equation used to eliminate the i'th var

45d4   inline void read(int _n, double _res[VAR], double _co[VAR][VAR])
  {
d763      rekt = false;
366f      n = _n;
97cb     for(int i = 0; i < n; i++){
0b33      u[i] = false;
21be      used[i] = 0;
58dd      ans[i] = 0.0;
eabb     }
da06     for(int i = 0; i < n; i++){
d7a1       res[i] = _res[i];
5389       for(int j = 0; j < n; j++)
543a         co[i][j] = _co[i][j];
d2dc     }
e6eb   }

4912   inline void run(){
403c     for(int i = 0; i < n; i++){
b8e0       used[i] = -1;
5c13       for(int j = 0; j < n; j++){
3b8b         if(u[j]) continue;
17c4         if(is_zero(co[j][i])) continue;
3681         used[i] = j;
8bb1         break;
8c20       }
5306       if(used[i] < 0){ //variable already eliminated from all
  the equations
5194         cout << "Linearly dependent equations found!" << endl;
7833         rekt = true;
701b         return;
dbfd       }
980e       u[used[i]] = 1;
be1d       for(int j = 0; j < n; j++){
4c9b         if(u[j]) continue;
5618         if(is_zero(co[j][i])) continue;
             //eliminating variable i from equation j
0f3f         double C = co[j][i] / co[used[i]][i];
b67e         for(int k = 0; k < n; k++){
76dc          co[j][k] -= C * co[used[i]][k];
e3cb         }
e04b         res[j] -= C * res[used[i]];
5f7f       }
fa86     }

76d8     for(int i = n - 1; i >= 0; i--){
79e0       for(int j = i + 1; j < n; j++){
fa0e         res[used[i]] -= co[used[i]][j] * ans[j];
93a0         co[used[i]][j] = 0;
04e0       }
4a43       res[used[i]] /= co[used[i]][i];
b056       co[used[i]][i] = 1.0F;
77f0       ans[i] = res[used[i]];
83c2     }
1463   }

9e44   inline void print(int case_no){
6722     if(rekt){
b212      cout << "Bug in code!\n";
495f      return;
bfea     };
e484     cout << "Case " << case_no << ": " << fixed << setprecision(10) << ans[0];
7257     cout << '\n';
bbef   }
13d6 };


0ed2 int main(){
eb0b  freopen("ioi.in", "r", stdin);
07af  cin >> t;
6168  for(int qq = 1; qq <= t; qq++){
642d      for(int i = 0; i < N; i++){
815f          for(int j = 0; j < N; j++){
35ff              coefficients[i][j] = 0.0;
f9d2          }
a90f          to[i] = i;
fd44          result[i] = 1.0;
2797      }
7be2      result[N - 1] = 0.0; // For last cell, equation is different.

aace      cin >> n;
f6d3      for(int i = 1; i <= n; i++){
a79a          cin >> u >> v;
8f09          u--, v--;
4a78          to[u] = v;
c22e      }

c3c4      double p = (1.0 / 6.0);
b152      for(int i = 0; i < N - 1; i++){
93f0          coefficients[i][i] = 1.0;
d8b6          for(int j = 1; j <= 6; j++){
9767              if(i + j < N)
b768                  coefficients[i][to[i + j]] -= p; // I go to (to[i + j])
08da              else
6c79                  coefficients[i][i] -= p; // I stay at the same place!
cb1e          }
f625      }
9ab8      coefficients[N - 1][N - 1] = 1.0;

e941      gaussian_elimination ge;
89f4      ge.read(N, result, coefficients);
a3f1      ge.run();
232c      ge.print(qq);
6164  }
2376 }
```

1.4 Matrix Exponentiation

1.4.1 MatrixExponentiation-hashed

```
     // Some Codeforces Educational Round
```

```
f425 #include "bits/stdc++.h"
3d1e using namespace std;


5d84 /*------ Matrix Exponentiation Template ------*/

a9d7 const int ORD = 2; // Order of Square Matrix
6f4e const int MOD = 1000000007; // Modulo

b331 inline int prod(int x, int y){
470b   long long res = x * 1LL * y;
40d8   if(res >= MOD) res %= MOD;
afe6   return res;
3502 }

bf30 inline int add(int x, int y){
3c05   int res = x + y;
c3e5   if(res >= MOD) res -= MOD;
cf29   return res;
4e04 }

2465 struct matrix{
afa6   int mat[ORD][ORD];
da65   matrix(){
eec6          for(int i = 0; i < ORD; i++)
3f43                 for(int j = 0; j < ORD; j++)
15b4                        mat[i][j] = 0;
a0d0   }
52d1   friend matrix operator * (matrix x, matrix y){
8201          matrix res;
dcea          for(int i = 0; i < ORD; i++)
295a                 for(int j = 0; j < ORD; j++)
712a                        for(int k = 0; k < ORD; k++)
b129                               res.mat[i][j] = add(res.mat[i][j],
    prod(x.mat[i][k], y.mat[k][j]));
d39a          return res;
d6e1   }
504d };

5e85 matrix base;

     // power(n) returns base ^ {n}
e51d matrix power(matrix cur, long long p){
be90   if(p == 1) return base;
83d4   matrix res = power(cur, p >> 1);
fe4d   res = res * res;
183b   if(p & 1) res = res * base;
79de   return res;
f3b3 }

7a6f /*------------ End of Template -------------*/


93a3 int a, b, x;
e63b long long n;

0ed2 int main(){
eb0b   freopen("ioi.in", "r", stdin);
3363   cin >> a >> b >> n >> x;
8e8c   base.mat[0][0] = a, base.mat[0][1] = 1;
c023   base.mat[1][0] = 0, base.mat[1][1] = 1;
9b76   matrix result = power(base, n);
5e94   cout << add(prod(result.mat[0][0], x), prod(result.mat[0][1], b))
    << ’\n’;
a289 }
```

2 DataStructures

2.1 2D Segment Tree

2.1.1 2DSegmentTree-hashed

```
8c39 #include <bits/stdc++.h>
3a00 using namespace std;

ad15 const int MAX = 275000;
928c int r, c, t;
1d81 vector < int > all_x, all_y;
2044 int type[MAX], a1[MAX], b1[MAX], a2[MAX], b2[MAX];
1bdc long long val[MAX];

37fa inline long long gcd(long long X, long long Y) {
5eef    long long tmp;
a4b9    while (X != Y and Y != 0) {
b779        tmp = X;
5cbe        X = Y;
b53c        Y = tmp % Y;
e5d4   }
2425   return X;
295c }

fb9e struct node{
bc2b  node *left, *right, *outer;
00f3  long long val;
5490  node(){
b6cd      left = right = outer = NULL;
9966      val = 0LL;
34b6  }
f5b2  inline void create(node* &x){
a5cb      if(!x) x = new node();
5d53  }
eb5e  inline node* update_y(node* &left_x, node* &right_x, int l, int r, int y, long long v, bool isLeaf){
41d8      if(l == r){
0248          if(isLeaf) val = v;
406d          else val = gcd((left_x) ? (left_x -> val) : (0), (right_x) ? (right_x -> val) : (0));
79dd          return this;
ebb3      }
0dec      int mid = (l + r) >> 1;
96db      if(mid >= y){
8d10          create(left);
fded          left = left -> update_y((left_x) ? (left_x -> left) : (left_x),
d933          (right_x) ? (right_x -> left) : (right_x), l, mid, y, v, isLeaf);
9eb4      }
3e29      else{
dff0          create(right);
44c2          right = right -> update_y((left_x) ? (left_x -> right) : (left_x),
915d          (right_x) ? (right_x -> right) : (right_x), mid + 1, r, y, v, isLeaf);
ecf7      }
9c81      val = gcd((left) ? (left -> val) : (0), (right) ? (right -> val) : (0));
a064      return this;
257e  }
abeb  inline node* update_x(int l, int r, int x, int y, long long v){
6dc5      create(outer), create(left), create(right);
2390      create(left -> outer), create(right -> outer);
07ae      if(l == r){
584e          outer = outer -> update_y(left -> outer, right -> outer, 1, c, y, v, 1);
3fed          return this;
6982      }
8fdd      int mid = (l + r) >> 1;
6f18      if(mid >= x) left = left -> update_x(l, mid, x, y, v);
1373      else right = right -> update_x(mid + 1, r, x, y, v);
1d39      outer = outer -> update_y(left -> outer, right -> outer, 1, c, y, v, 0);
d167      return this;
3ef6  }
85cd  inline long long query_y(int l, int r, int b1, int b2){
7588      if(l > b2 or r < b1) return 0;
b75b      if(l >= b1 and r <= b2) return val;
5104      int mid = (l + r) >> 1;
402a      return gcd((left) ? (left -> query_y(l, mid, b1, b2)) : (0),
be48             (right) ? (right -> query_y(mid + 1, r, b1, b2)) : (0));
458f  }
443c  inline long long query_x(int l, int r, int a1, int b1, int a2, int b2){
b24c      if(l > a2 or r < a1) return 0;
ec90      if(l >= a1 and r <= a2) return (outer) ? (outer -> query_y(1, c, b1, b2)) : (0);
0acf      int mid = (l + r) >> 1;
0160      return gcd((left) ? (left -> query_x(l, mid, a1, b1, a2, b2)) : (0),
2525               (right) ? (right -> query_x(mid + 1, r, a1, b1, a2, b2)) : (0));
2954  }
bd32 };

77af inline int compressX(int x){
f8b4  return lower_bound(all_x.begin(), all_x.end(), x) - all_x.begin() + 1;
a7b8 }

57ef inline int compressY(int y){
b3f0  return lower_bound(all_y.begin(), all_y.end(), y) - all_y.begin() + 1;
85e2 }

6a43 node* root = new node();
d442 int main(){
```

```
e781    scanf("%d %d %d\n", &r, &c, &t);
511d    for(int i = 1; i <= t; i++){
10c7            scanf("%d ", &type[i]);
aba2            if(type[i] == 1){
f00a                    scanf("%d %d %lld\n", &a1[i], &b1[i], &val[i]);
46cb                    a1[i]++, b1[i]++;
2c5d                    all_x.push_back(a1[i]);
e143                    all_y.push_back(b1[i]);
1f77            }
dfe9            else{
aa6f                    scanf("%d %d %d %d\n", &a1[i], &b1[i], &a2[i],
     &b2[i]);
a674                    a1[i]++, b1[i]++, a2[i]++, b2[i]++;
ff0f                    all_x.push_back(a1[i]), all_x.push_back(a2[i]);
922f                    all_y.push_back(b1[i]), all_y.push_back(b2[i]);
7cec            }
639a    }

bcb2    sort(all_x.begin(), all_x.end());
0104    sort(all_y.begin(), all_y.end());
a972    all_x.resize(unique(all_x.begin(), all_x.end()) - all_x.begin());
135b    all_y.resize(unique(all_y.begin(), all_y.end()) - all_y.begin());
baa0    r = all_x.size(), c = all_y.size();

91ee    for(int i = 1; i <= t; i++){
8e92            if(type[i] == 1){
edfc                    a1[i] = compressX(a1[i]), b1[i] = compressY(b1[i]);
5c97                    root = root -> update_x(1, r, a1[i], b1[i], val[i]);
ba99            }
28bb            else{
4a64                    a1[i] = compressX(a1[i]), a2[i] = compressX(a2[i]);
36cf                    b1[i] = compressY(b1[i]), b2[i] = compressY(b2[i]);
ae98                    printf("%lld\n", root -> query_x(1, r, a1[i],
     b1[i], a2[i], b2[i]));
c509            }
4e55    }
aa0f  }
```

2.2 HLD Trick

2.2.1 TreePairs-hashed

```
     // Find number of pairs (u, v) such that A[u] * A[v] = A[lca(u, v)]


f425 #include "bits/stdc++.h"
3d1e using namespace std;

622c const int N = 1e5 + 50;

41cc int n, arr[N], par[N];
7fe0 vector < int > adj[N];
ed39 map < int, int > val[N];
0f80 long long ans = 0;

62f3 inline int root(int x){
e4ba     if(par[x] == x) return x;
111b     return par[x] = root(par[x]);
d8f5 }

839e inline void unite(int u, int v, int target){
a23e     u = root(u), v = root(v);
b3bb     if((int) val[u].size() < (int) val[v].size()){
998c         for(map < int, int > :: iterator it = val[u].begin(); it !=
     val[u].end(); it++){
1a43             int cur = (*it).first;
0514             if(target % cur == 0) ans += ((*it).second * 1LL *
     val[v][target / cur]);
a055         }
be50         for(map < int, int > :: iterator it = val[u].begin(); it !=
     val[u].end(); it++){
55fb             int cur = (*it).first;
8e00             val[v][cur] += (*it).second;
040d         }
0440         val[u].clear();
b181         par[u] = v;
0df1     }
9ce0     else{
d33a         for(map < int, int > :: iterator it = val[v].begin(); it !=
     val[v].end(); it++){
8f2f             int cur = (*it).first;
53bd             if(target % cur == 0) ans += ((*it).second * 1LL *
     val[u][target / cur]);
eae0         }
3f3a         for(map < int, int > :: iterator it = val[v].begin(); it !=
     val[v].end(); it++){
572e             int cur = (*it).first;
cf35             val[u][cur] += (*it).second;
ae04         }
8575         val[v].clear();
902d         par[v] = u;
6cfc     }
e31a }

c559 inline void dfs(int u, int p){
07b9     val[u][arr[u]]++;
3da3     for(int i = 0; i < (int) adj[u].size(); i++){
853d         int v = adj[u][i];
2528         if(v == p) continue;
ada4         dfs(v, u);
ac3f         unite(u, v, arr[u]);
fd1c     }
e795 }

0ed2 int main(){
e312     freopen("inp.in", "r", stdin);
7445     scanf("%d", &n);
73e9     for(int i = 1; i < n; i++){
e30c         int u, v;
f63a         scanf("%d %d", &u, &v);
78f7         adj[u].push_back(v);
a0bb         adj[v].push_back(u);
dd78     }
2de8     for(int i = 1; i <= n; i++){
bb7a         scanf("%d", arr + i);
7055         par[i] = i;
abff     }
73d3     dfs(1, -1);
3872     printf("%lld\n", ans);
91be }
```

2.3 Persistent Segment Tree

2.3.1 PersistentSegmentTrees-hashed

```
782b /*
2313         WCIPEG Problem
0264   Prints sum of K maximum sum subarrays, each of L <= length <= R
f905   Array has negative elements as well.
9563   Add f[i]th best subarray starting at index (i) of valid length for each (i)
2530   into a priority queue. Initially, let f[i] = 1 for all (i).
5f90   Pop the best value from the priority queue k times, increment f[i] each time
6e46   and add a new value to it after each pop.
c1b5 */

8c39 #include <bits/stdc++.h>
3a00 using namespace std;

d689 const int MAX = 500005;
0885 const int INF = 1000000000;

fb9e struct node{
770f   node *lc, *rc;
e0d6   int val;
b4ea   node(node *x = NULL, node *y = NULL, int v = 0){
e0e8           lc = x, rc = y, val = v;
477a   }
e941   inline void create(node *&x){
990c           if(!x) x = new node();
64b5   }
4e3f   inline int sum(node *x){
c9f8           return (x) ? (x -> val) : (0);
c632   }
dab4   inline node *insert(int l, int r, int value){
5f91           node *nw = new node();
e7a5           if(l == r){
01bb                   nw -> val = val + 1;
67e7                   return nw;
3b42           }
dd1d           int mid = (l + r) >> 1;
a2ba           if(mid >= value){
71e7                   nw -> rc = rc;
ab17                   create(lc);
6927                   nw -> lc = lc -> insert(l, mid, value);
3b34           }
fe7b           else{
2e57                   nw -> lc = lc;
cb94                   create(rc);
720a                   nw -> rc = rc -> insert(mid + 1, r, value);
53ed           }
d8c5           nw -> val = sum(nw -> lc) + sum(nw -> rc);
ac11           return nw;
8d1d   }
3fed   inline int query(node *r1, node *r2, int l, int r, int k){
1c46       if(l == r) return r;
cc31       int goRight = sum(r1 -> rc) - sum(r2 -> rc);
2a6e       int mid = (l + r) >> 1;
e793       if(goRight >= k){
6928           create(r1 -> rc), create(r2 -> rc);
9da1           return query(r1 -> rc, r2 -> rc, mid + 1, r, k);
0c90       }
2c60       else{
a1bb           create(r1 -> lc), create(r2 -> lc);
6432           return query(r1 -> lc, r2 -> lc, l, mid, k - goRight);
935c       }
e4e7   }
d1c1 };

8e4e node *root[MAX], *dummy;
13cd map < int, int > compress;
2298 int n, k, l, r, lim;
f162 int arr[MAX], f[MAX], original[MAX];

02f2 inline int get(int i, int j){
0f15   if(i + l - 1 > n || j > r - l + 1) return -INF;
de4f   return original[dummy -> query(root[min(i + r - 1, n)], root[i + l - 2], 1, lim, j)] - arr[i - 1];
7e8f }

0ed2 int main(){
be81   scanf("%d %d %d %d", &n, &k, &l, &r);
d426   compress[-INF];
82ec   for(int i = 1; i <= n; i++){
7a51       scanf("%d", arr + i);
c9ce       arr[i] += arr[i - 1];
cd3a       compress[arr[i]];
d614   }
4d4a   for(auto &it : compress) it.second = ++lim;
05fe   for(auto it : compress) original[it.second] = it.first;
7261   root[0] = dummy = new node();
a13f   for(int i = 1; i <= n; i++){
d7d9       root[i] = root[i - 1] -> insert(1, lim, compress[arr[i]]);
cec3   }
5256   priority_queue < pair < int, int > > sums;
baaf   for(int i = 1; i <= n; i++){
8353       sums.push({get(i, ++f[i]), i});
9c67   }
cfc5   long long res = 0;
473d   while(k--){
d73f       res += (sums.top().first);
6938       sums.push({get(sums.top().second, ++f[sums.top().second]), sums.top().second});
340c       sums.pop();
61dd   }
0f6c   printf("%lld\n", res);
6006 }
```

2.4 Sparse Table

2.4.1 SparseTable-hashed

```
83b0 int log_table[N], mx[LN][N], mn[LN][N];
0f23 inline void preprocess(){
3ee8   log_table[1] = 0;
761b   for(int i = 2; i <= n; i++) log_table[i] = log_table[i >> 1] + 1;

7d9c   for(int i = 1; i <= n; i++) mx[0][i] = a[i];
e9af   for(int i = 1; i < LN; i++)
8bcf       for(int j = 1; j + (1 << i) - 1 <= n; j++)
cd12           mx[i][j] = max(mx[i - 1][j] , mx[i - 1][j + (1 << (i - 1))]);

4cfc   for(int i = 1; i <= n; i++) mn[0][i] = b[i];
eab9   for(int i = 1; i < LN; i++)
8a44       for(int j = 1; j + (1 << i) - 1 <= n; j++)
7743           mn[i][j] = min(mn[i - 1][j] , mn[i - 1][j + (1 << (i - 1))]);
1bc7 }

4329 inline int get_max(int l, int r){
4bfa   int k = log_table[r - l + 1];
8c0c   return max(mx[k][l] , mx[k][r - (1 << k) + 1]);
641d }

424d inline int get_min(int l, int r){
4e6a   int k = log_table[r - l + 1];
8a42   return min(mn[k][l] , mn[k][r - (1 << k) + 1]);
142f }
```

2.5 Treaps

2.5.1 ImplicitTreaps-hashed

```
f425 #include "bits/stdc++.h"
3d1e using namespace std;

8585 const int NMAX = 40010;

fb9e struct node{
9c03   int left, right, pr, sz, rev;
e8df };

3a24 node tree[NMAX];
0116 int N, null, root;

a777 inline int create_node(){
19c4   tree[N].pr = rand();
9667   tree[N].sz = 1;
ea76   tree[N].left = tree[N].right = null;
5c51   tree[N].rev = 0;
3b3e   return N++;
f1a4 }

f4a1 inline int upd(int x){
743e   int l = tree[x].left, r = tree[x].right;
cb37   tree[x].sz = tree[l].sz + tree[r].sz + 1;
c70a   return x;
5645 }

     // Swap left child and right child if it needs to be reversed.

cc56 inline void down(int rt){
17f2   if(!tree[rt].rev) return;
cf7a   swap(tree[rt].left, tree[rt].right);
0294   tree[rt].rev = 0;
37bf   tree[tree[rt].left].rev ^= 1, tree[tree[rt].right].rev ^= 1;
f9c0 }


782b /*
2a6b   Takes the treap rooted at "rt" and puts the k smallest elements
c495   in it into sp.first, and the rest into sp.second
755f */
4d38 inline pair < int, int > split(int rt, int k){
5eba   if(rt >= null) return make_pair(null, null);
92df   down(rt);
ba82   pair < int, int > sp;
e64b   if(tree[tree[rt].left].sz >= k){
f96a          sp = split(tree[rt].left, k);
25fc          tree[rt].left = sp.second;
8e90          sp.second = upd(rt);
21fb          return sp;
d972   }
dd0a   else{
ba4e          k -= tree[tree[rt].left].sz;
1b01          sp = split(tree[rt].right, k - 1);
a8ef          tree[rt].right = sp.first;
9139          sp.first = upd(rt);
dcb3          return sp;
9e98   }
c489 }

     // Standard Treap Merge : down() is called to initiate reverse when
        needed.

155a inline int merge(int l, int r){
131b   if(l >= null) return r;
ec59   if(r >= null) return l;
33ae   if(tree[l].pr > tree[r].pr){
a76f          down(l);
c2d8          tree[l].right = merge(tree[l].right, r);
374c          return upd(l);
61c7   }
87d6   else{
d85b          down(r);
65c1          tree[r].left = merge(l, tree[r].left);
e340          return upd(r);
0767   }
3846 }


782b /*
0885   Returns the index (node no. in treap) of the (k + 1)th smallest
    value
732f   in the treap. In this problem, index equals value so printing the
    index
f957   suffices. However, if array values are different, then you should
    maintain
```

```
3d3c   a parameter 'val' in each treap node and print treap[idx].val.

d869   Note that this is an implicit treap, hence here we are simply returning
0e1c   the (k + 1)th value in the array, since the treap is ordered based on array
1e16   indices.
d5a9 */

bc0a inline int search(int rt, int k){
5728   if(rt >= null) return rt;
de4f   down(rt);
71aa   if(tree[tree[rt].left].sz > k){
3686           return search(tree[rt].left, k);
31c9   }
80fe   else{
ad33           k -= tree[tree[rt].left].sz;
f6aa           if(!k) return rt;
2ee2           return search(tree[rt].right, k - 1);
110a   }
50f5 }

782b /*
6772   Suppose array[1..N] is present. reverse(i, j) takes subarray [i...j] of it (1 based)
00b0   and reverses it.
7c2e */

0627 inline void reverse(int i, int j){
5e3a   pair < int, int > sp, sp2;
78d6   sp  = split(root, j); // sp.first = arr[1..j], sp.second = arr[j + 1...N]
2529   sp2 = split(sp.first, i - 1); // sp2.first = arr[1..i - 1], sp2.second = arr[i..j]
d56b   tree[sp2.second].rev = 1; // sp2.second needs to be reversed, mark it.
dbad   sp.first = merge(sp2.first,sp2.second); // Now merge everything normally!
1b96   assert(merge(sp.first,sp.second) == root); // Merge
b0a1 }

782b /*
d941     Insert element at position (i + 1) in the array i.e. after position (i)
3fbf     Here element value is not inputted since it's equal to index.
af9a     Look at other codes for utilising this function
b685 */

663d inline int insert(int i){
c9ad   pair < int, int > sp = split(root, i);
bd00   int x = create_node();
dc38   sp.first = merge(sp.first, x);
dacf   return merge(sp.first, sp.second);
7eab }

0ed2 int main(){
5123   int i, j, n;
3273   scanf("%d", &n);
1777   null = 40001;
cdc4   root = null;
c0a3   while(n--) root = insert(N);
6015   while(true){
14b5           scanf("%d",&n);
affa           if(n >= 2) break;
b8c6           if(n){
5a86                   scanf("%d %d", &i, &j);
a628                   reverse(i, j);
454c           }
c244           else{
5f41                   scanf("%d", &i);
af4b                   int ans = search(root, i - 1);
4a4f                   printf("%d\n", ans + 1);
7a2f           }
7bac   }
63a0 }
```

2.5.2 TreapBST-hashed

```
782b /*
3e5d         SPOJ RaceTime
878d   1) Update A[i] = X for given i and X
a414   2) Print # of i such that L <= i <= R and A[i] <= X, for given L, R and X
5507 */

8c39 #include <bits/stdc++.h>
c980 #define pii pair < int, int >
5451 using namespace std;
```

```
4cca const int MAXN = 100005;
cdb0 const int MAXQ = 50005;
b852 const int LN   = 20;
e1b0 const int EMPTY = (MAXN + MAXQ) * LN - 1;

ceec int N, n, q, arr[MAXN];
2929 int treap_roots[MAXN];

1199 struct treap_node{
b162   int val, pri, siz, lc, rc;
3022 }treap[(MAXN + MAXQ) * LN];

943d inline int create_node(int val){
2593   N = N + 1;
c1cf   treap[N].val = val;
aad7   treap[N].pri = rand();
8ce2   treap[N].siz = 1;
8b96   treap[N].lc = treap[N].rc = EMPTY;
d64b   return N;
5ecf }

275a inline void refresh(int root){
de1a   treap[root].siz = treap[treap[root].lc].siz + 1 +
    treap[treap[root].rc].siz;
d68d }

782b /*
7f0f   splits treap into two treaps parts.first and parts.second such that
bd9a   parts.first comprises all elements with val <= key and
    parts.second comprises
23b6   all elements with val > key.
bda6 */

5f9b inline pii split(int root, int key){
ede2   pii parts = pii(EMPTY, EMPTY);
59d9   if(root == EMPTY) return parts;
f0b5   if(treap[root].val <= key){
80f2           parts = split(treap[root].rc, key);
f283           treap[root].rc = parts.first;
a9d8           refresh(root);
4f3b           parts.first = root;
cd61           return parts;
0e16   }
6f61   else{
df25           parts = split(treap[root].lc, key);
09f5           treap[root].lc = parts.second;
7235           refresh(root);
df9b           parts.second = root;
c768           return parts;
4646   }
324f }

782b /*
7bbf   Merge treaps l, r.
7c09   Note largest key in l must be <= smallest key in r
5271 */

155a inline int merge(int l, int r){
a424   if(l == EMPTY) return r;
bf2d   if(r == EMPTY) return l;
6525   if(treap[l].pri > treap[r].pri){
850c           treap[l].rc = merge(treap[l].rc, r);
5748           refresh(l);
1896           return l;
b0b9   }
38be   else{
073a           treap[r].lc = merge(l, treap[r].lc);
3dcc           refresh(r);
6cfc           return r;
e31a   }
d765 }

782b /*
1967   Insert treap_node named 'add' with value 'treap[add].val' into
6514   treap rooted at 'root'
1537 */

a19b inline int insert(int root, int add){
55e0   if(root == EMPTY) return add;
3061   pii parts = split(root, treap[add].val - 1);
58fc   return merge(merge(parts.first, add), parts.second);
e2ba }

782b /*
6978   Remove 'rem_value' from treap rooted at 'root'
0e34 */

38be inline int erase(int root, int rem_value){
62b4   if(root == EMPTY) return EMPTY;
9a0a   if(treap[root].val == rem_value){
9ae3          return merge(treap[root].lc, treap[root].rc);
1caa   }
0620   if(treap[root].val > rem_value){
59f8          treap[root].lc = erase(treap[root].lc, rem_value);
4775          refresh(root);
105d          return root;
e8ff   }
1b92   else{
e834          treap[root].rc = erase(treap[root].rc, rem_value);
75b2          refresh(root);
9e38          return root;
c48c   }
6659 }

782b /*
f74a   Returns # of elements in the treap rooted at 'root' that
161e   has a value <= k.
d7ab */

ce33 inline int query_k(int root, int k){
e47f   if(root == EMPTY) return 0;
8957   if(treap[root].val <= k){
c939          return treap[treap[root].lc].siz + 1 +
    query_k(treap[root].rc, k);
ce34   }
7e01   else{
1c02          return query_k(treap[root].lc, k);
109d   }
e8f9 }

782b /*
f4d2   Maintain a BIT in which each node is a TREAP.
6705   treap_roots[] denotes the roots of the treaps.
9177 */

57bb inline void update(int idx, int val, int type){
22b9   for(int i = idx; i <= n; i += i & -i){
87e8          if(type) treap_roots[i] = insert(treap_roots[i],
    create_node(val));
a147          else    treap_roots[i] = erase(treap_roots[i], val);
3d77   }
b996 }

4851 inline int query(int idx, int k){
0ccd   int res = 0;
9120   for(int i = idx; i > 0; i -= i & -i){
c53f          res += query_k(treap_roots[i], k);
fe54   }
8045   return res;
2c7f }

0ed2 int main(){
14a7   scanf("%d %d", &n, &q);
c20c   for(int i = 1; i <= n; i++){
deb1          treap_roots[i] = EMPTY;
8e88   }
d5c1   for(int i = 1; i <= n; i++){
b104          scanf("%d", arr + i);
e37b          update(i, arr[i], 1);
df66   }
9029   char buf[1];
3f83   while(q--){
d4f8          scanf("%s", buf);
9071          if(buf[0] == 'M'){
8f47                 int i, x;
4d7e                 scanf("%d %d", &i, &x);
1bce                 update(i, arr[i], 0);
8cf2                 arr[i] = x;
9897                 update(i, arr[i], 1);
bcb9          }
38b8          else{
ab27                 int st, en, x;
7cc2                 scanf("%d %d %d", &st, &en, &x);
ff49                 printf("%d\n", query(en, x) - query(st - 1, x));
4f87          }
3a01   }
09ad }
```

2.6 Trie

2.6.1 Trie-hashed

```
8c39 #include <bits/stdc++.h>
3a00 using namespace std;

9431 int n, k, x;
```

```
782b /*
4b60   A subarray of a[] is beautiful if the bitwise xor of all the
    elements in the subarray
8bc9   is at least k. Print count of such subarrays.
a24c */

fb9e struct node{
770f   node *lc, *rc;
9227   int leaves;
eff3   node(node *_lc = NULL, node *_rc = NULL, int _leaves = 0){
9e4b          lc = _lc;
9004          rc = _lc;
e699          leaves = _leaves;
cf49   }
ba67   inline int val(node *x){
eed0          return x ? x -> leaves : 0;
870b   }
995d   inline void create(node* &x){
9e10          if(!x) x = new node();
848d   }
f8f4   inline int query(int pos, int prefix){
482a          if(pos == -1) return 0;
e2d2          int k_bit = k & (1 << pos);
1d5d          int p_bit = prefix & (1 << pos);
a4af          int res = 0;
4fe5          if(!k_bit){
da5f                 if(!p_bit){
fbdd                        res += val(rc);
de03                        create(lc);
bb77                        res += lc -> query(pos - 1, prefix);
bda6                 }
b738                 else{
4d52                        res += val(lc);
c152                        create(rc);
f4dc                        res += rc -> query(pos - 1, prefix);
e7db                 }
df43          }
c589          else{
4fc1                 if(p_bit){
e76b                        create(lc);
a7c3                        res += lc -> query(pos - 1, prefix);
1d43                 }
c5e8                 else{
b443                        create(rc);
4e54                        res += rc -> query(pos - 1, prefix);
a20f                 }
7d6d          }
e08b          return res;
5f79   }
1504   node *insert(int pos, long long prefix){
183a          if(pos == -1){
2877                 ++leaves;
4d0d                 return this;
6a15          }
b90c          ++leaves;
c3af          int p_bit = prefix & (1 << pos);
4f47          if(!p_bit){
eb6a                 create(lc);
3eee                 lc = lc -> insert(pos - 1, prefix);
718a          }
a15e          else{
d88a                 create(rc);
22ed                 rc = rc -> insert(pos - 1, prefix);
696a          }
778f          return this;
7bc1   }
1866 };

d02d node *trie = new node();

0ed2 int main(){
eb0b   freopen("ioi.in", "r", stdin);
8c7a   scanf("%d %d", &n, &k);
cb6b   k--;
fe99   long long res = 0;
c040   int prefix_xor = 0;
68a9   for(int i = 0; i < n; i++){
06b4          scanf("%d", &x);
dffb          prefix_xor ^= x;
96df          res += trie -> query(30, prefix_xor) + (prefix_xor > k);
327d          trie = trie -> insert(30, prefix_xor);
e9ee   }
16a8   printf("%lld\n", res);
40c8 }
```

2.7 Wavelet Tree

2.7.1 WaveletTree-hashed

```
8c39 #include <bits/stdc++.h>
3a00 using namespace std;

fb7a typedef vector < int > :: iterator iter;


782b /*
a868     ------------ Wavelet Tree Template ----------

6442     quantile(k, a, b) : k'th smallest element in [a, b)
486e     range(x, y, a, b) : # of elements with value in range [x, y] in
     subarray [a, b)
baf8     rank(x, k) : # of occurrences of x in [0, k)
3fa4     push_back(x) : Append another value x to the existing array.
4c6e                    Note : x should be in [0, sigma)
f06a     pop_back() :  Pop the last element from the existing array.
62a4     swap_adj(i) : Swap arr[i] and arr[i + 1]. Assumes i is in [0, n
     - 1)

9d0f     WaveTree obj(arr, sigma) : Creates a Wavelet Tree on the vector
     'arr', alphabet size [0, sigma)

42b9     All indices are Zero-Based.

05a0     ----------------------------------------------
382f */


b2f8 class WaveTree {
55c7     vector < vector < int > > tree;
81a3     vector < int > arr_copy;
         // tree[u][i] = uptil index (i) in node (u), how many values are
             <= (mid)
e547     int n, s;
         // O(n * log (sigma)) construction
3ae3     inline void build(iter b, iter e, int l, int r, int u) {
2e2a         if (l == r) return;
5c6a         int m = (l + r) / 2;
885b         tree[u].reserve(e - b + 1);
d3b1         tree[u].push_back(0);
054a         for (iter it = b; it != e; ++it)
bb42             tree[u].push_back(tree[u].back() + (*it <= m));
2063         iter p = stable_partition(b, e, [=](int i){ return i <= m;});
             // arr[b, p) have elements <= m and arr[p, e) have > m
9d7b         build(b, p, l, m, u * 2);
83dd         build(p, e, m + 1, r, u * 2 + 1);
ec63     }

b868     int qq, w;
e9e3     inline int range(int a, int b, int l, int r, int u) {
c29b         if (r < qq or w < l) return 0;
2b0e         if (qq <= l and r <= w) return b - a;
3909         int m = (l + r) / 2, za = tree[u][a], zb = tree[u][b];
8562         return range(za, zb, l, m, u * 2) +
d94d                 range(a - za, b - zb, m + 1, r, u * 2 + 1);
6eb7     }

19e7 public:

         //arr[i] in [0, sigma)
9989     WaveTree(vector < int > arr, int sigma) {
9612         n = arr.size();
c85c         s = sigma;
1156         tree.resize(s * 2);
8a6b         arr_copy = arr;
094c         build(arr.begin(), arr.end(), 0, s - 1, 1);
6037     }

         //k in [1, n], [a, b) is 0-indexed, -1 if error
5aad     inline int quantile(int k, int a, int b) {
2054         if (a < 0 or b > n or k < 1 or k > b - a) return -1;
eb45         int l = 0, r = s - 1, u = 1, m, za, zb;
e722         while (l != r) {
283a             m = (l + r) / 2;
492a             za = tree[u][a];
085e             zb = tree[u][b];
e02d             u *= 2;
b182             if (k <= zb - za)
692f                 a = za, b = zb, r = m;
753e             else
a99e                 k -= zb - za, a -= za, b -= zb,
4e8a                 l = m + 1, ++u;
5209         }
a993         return r;
9d31     }

         //Counts numbers in [x, y] in positions [a, b)
a09d     inline int range(int x, int y, int a, int b) {
0bf9         if (y < x or b <= a) return 0;
45a7            qq = x; w = y;
44a5            return range(a, b, 0, s - 1, 1);
2a58        }

        //Count occurrences of x in positions [0, k)
7b01    inline int rank(int x, int k) {
f0a9        int l = 0, r = s - 1, u = 1, m, z;
59e3        while (l != r) {
478a            m = (l + r) / 2;
8142            z = tree[u][k];
6e69            u *= 2;
14dc            if(x <= m) k = z, r = m;
6309            else k -= z, l = m + 1, ++u;
4b65        }
0d8a        return k;
5011    }

        //x in [0, sigma)
38bd    inline void push_back(int x) {
7e88        int l = 0, r = s - 1, u = 1, m, p;
fd46        ++n;
8713        while (l != r) {
7036            m = (l + r)/2;
fe17            p = (x <= m);
606e            tree[u].push_back(tree[u].back() + p);
f819            u *= 2;
a6fd            if(p) r = m;
329d            else l = m + 1, ++u;
e9e9        }
4f32    }

        //Assumes that array is non-empty
897c    inline void pop_back() {
c9df        int l = 0, r = s - 1, u = 1, m, p, k;
87bf        --n;
2884        while (l != r) {
9bd3            m = (l + r) / 2;
0e7d            k = tree[u].size();
756d            p = tree[u][k - 1] - tree[u][k - 2];
0e7a            tree[u].pop_back();
f22e            u *= 2;
f745            if(p) r = m;
d3db            else l = m + 1, ++u;
dee3        }
1e8a    }
```

```
         //swap arr[i] with arr[i + 1], i in [0, n - 1)
de57     inline void swap_adj(int i){
c732          int &x = arr_copy[i], &y = arr_copy[i + 1];
2944          int l = 0, r = s - 1, u = 1;
c73e          while(l != r){
be75            int m = (l + r) / 2, p = (x <= m), q = (y <= m);
2643            if (p != q){
d6b7                tree[u][i + 1] ^= tree[u][i] ^ tree[u][i + 2];
0b6a                break;
5026            }
473a            int z = tree[u][i];
520a            u *= 2;
7378            if(p) i = z, r = m;
a441            else i -= z, l = m + 1, ++u;
0d5f        }
5426        swap(x, y);
32dc     }
5f34 };

0ed2 int main() {

f547     int n, q;
4b1e     scanf("%d %d", &n, &q);
e886     vector < int > arr(n);
3277     for(int i = 0; i < n; i++) scanf("%d", &arr[i]);

         //Co-ordinate Compression
14b1     vector < int > values;
75e6     for(int i = 0; i < n; i++){
ef5e         values.push_back(arr[i]);
f707     }

217c     sort(values.begin(), values.end());
f3b3     values.resize(unique(values.begin(), values.end()) -
    values.begin());

3afe     int sigma = 0;
6876     vector < int > orig(n);
022a     for(int i = 0; i < n; i++){
892e         int init = arr[i];
4d45         arr[i] = lower_bound(values.begin(), values.end(), arr[i]) -
    values.begin();
a4a7         orig[arr[i]] = init;
3dbb         sigma = max(sigma, arr[i]);
d990    }

782b    /*
277a        1) Vector 'arr' represents the array
8135        2) 'sigma' represents the alphabet size i.e [0, sigma + 1)
    in this case.
1d4e    */

3364    WaveTree wt(arr, sigma + 1);

765a    for(int qq = 0; qq < q; qq++){
43d6        int cmd, i, k;
bb29        scanf("%d", &cmd);
7d88        if(cmd){
a6d6          scanf("%d", &i);
70b9          wt.swap_adj(i);
cbf8        }
9803        else{
08ca          scanf("%d %d", &i, &k);
              // val = 'k'th smallest element in [0, i + 1)
3e46          int val = orig[wt.quantile(k, 0, i + 1)];
91c7          printf("%d\n", val);
3cf3      }
999a    }
d4b1 }
```

3 Flows

3.1 BipartiteMatching-hashed

```
     // LIGHTOJ
f425 #include "bits/stdc++.h"
3d1e using namespace std;

aa6c const int N = 1005;
de44 const int M = 10005;

313b int t, n, m;

782b /*
a7ca   Hopcroft Karp Max Matching in O(E * sqrt(V))
4002   N = Number of Nodes, M = Number of Edges
1e9e   n1 = Size of left partite, n2 = Size of right partite
6d6a   Nodes are numbered from [0, n1 - 1] and [0, n2 - 1]

4cd8   init(n1, n2) declares the two partite sizes and resets arrays
2943   addEdge(x, y) adds an edge between x in left partite and y in
    right partite
0a4f   maxMatching() returns the maximum matching

3bae   Maximum Matching = Minimum Vertex Cover (Konig's Theorem)
8a2b   N - Maximum Matching = Maximal Independent Set
dacc */


089a int n1, n2, edges, last[N], previous[M], head[M];
0b17 int matching[N], dist[N], Q[N];
0019 bool used[N], vis[N];

84dc inline void init(int _n1, int _n2) {
caa4   n1 = _n1;
daeb   n2 = _n2;
a65c   edges = 0;
b185   fill(last, last + n1, -1);
2df1 }

d1fc inline void addEdge(int u, int v) {
6c37   head[edges] = v;
7659   previous[edges] = last[u];
2152   last[u] = edges++;
9177 }

7129 inline void bfs() {
fd58   fill(dist, dist + n1, -1);
05e4   int sizeQ = 0;
3798   for(int u = 0; u < n1; ++u){
1911          if(!used[u]){
25c4                 Q[sizeQ++] = u;
f0c8                 dist[u] = 0;
47fb          }
da42   }
6f0b   for(int i = 0; i < sizeQ; i++){
7d34          int u1 = Q[i];
07f5          for(int e = last[u1]; e >= 0; e = previous[e]){
f729                 int u2 = matching[head[e]];
d387                 if(u2 >= 0 && dist[u2] < 0){
069f                        dist[u2] = dist[u1] + 1;
516b              Q[sizeQ++] = u2;
5af6          }
b2aa      }
55e8    }
42d2 }

5b8e inline bool dfs(int u1) {
050a   vis[u1] = true;
e70c   for(int e = last[u1]; e >= 0; e = previous[e]){
176b       int v = head[e];
02f8       int u2 = matching[v];
8af5       if(u2 < 0 || !vis[u2] && dist[u2] == dist[u1] + 1 && dfs(u2)){
59f7           matching[v] = u1;
adc5           used[u1] = true;
9f32           return true;
9484       }
24d9   }
c9bb   return false;
de30 }

aef8 inline int maxMatching() {
ff22   fill(used, used + n1, false);
0d7d   fill(matching, matching + n2, -1);
6531   for(int res = 0; ;){
871f       bfs();
3379       fill(vis, vis + n1, false);
df7a       int f = 0;
11cf       for(int u = 0; u < n1; ++u)
a9e3           if(!used[u] && dfs(u))
0030               ++f;
a4cf       if(!f) return res;
c8f8       res += f;
c63a   }
d64c }

     // End of Hopcroft Karp Template


782b /*

4d84   Given a DAG with edges (u, v) print minimum path cover of it.
b8e2   Note : the paths should be vertex disjoint.

1170   Answer = n - maxMatching()
e8a7   No need to do transitive closure as vertex-disjoint is required!
b172   If vertex disjoint paths are not required, you need to perform a
4f6c   transitive closure of the DAG.
8b3d */


0ed2 int main(){
eb0b   freopen("ioi.in", "r", stdin);
a454   ios :: sync_with_stdio(false);
ee44   cin >> t;
8a81   for(int qq = 1; qq <= t; qq++){
508f       cin >> n >> m;
3f52       init(n, n);
3895       for(int i = 1; i <= m; i++){
6a63           int u, v;
80b6           cin >> u >> v;
c638           addEdge(u - 1, v - 1);
c64c       }
b7ca       cout << "Case " << qq << ": " << (n - maxMatching()) << '\n';
55c3   }
1ad3 }
```

3.2 Dilworths-hashed

```
     // LIGHTOJ
f425 #include "bits/stdc++.h"
3d1e using namespace std;

62fb const int N = 105;
bcb6 const int M = 105 * 105;

0d73 int t, n;
cd96 int arr[N];
7102 vector < int > values;

782b /*
a7ca   Hopcroft Karp Max Matching in O(E * sqrt(V))
4002   N = Number of Nodes, M = Number of Edges
1e9e   n1 = Size of left partite, n2 = Size of right partite
6d6a   Nodes are numbered from [0, n1 - 1] and [0, n2 - 1]

4cd8   init(n1, n2) declares the two partite sizes and resets arrays
```

```
2943   addEdge(x, y) adds an edge between x in left partite and y in
    right partite
0a4f   maxMatching() returns the maximum matching

3bae   Maximum Matching = Minimum Vertex Cover (Konig's Theorem)
8a2b   N - Maximum Matching = Maximal Independent Set
dacc */

089a int n1, n2, edges, last[N], previous[M], head[M];
0b17 int matching[N], dist[N], Q[N];
0019 bool used[N], vis[N];

84dc inline void init(int _n1, int _n2) {
caa4   n1 = _n1;
daeb   n2 = _n2;
a65c   edges = 0;
b185   fill(last, last + n1, -1);
2df1 }

d1fc inline void addEdge(int u, int v) {
6c37   head[edges] = v;
7659   previous[edges] = last[u];
2152   last[u] = edges++;
9177 }

7129 inline void bfs() {
fd58   fill(dist, dist + n1, -1);
05e4   int sizeQ = 0;
3798   for(int u = 0; u < n1; ++u){
1911          if(!used[u]){
25c4                 Q[sizeQ++] = u;
f0c8                 dist[u] = 0;
47fb          }
da42   }
6f0b   for(int i = 0; i < sizeQ; i++){
7d34          int u1 = Q[i];
07f5          for(int e = last[u1]; e >= 0; e = previous[e]){
f729                 int u2 = matching[head[e]];
d387                 if(u2 >= 0 && dist[u2] < 0){
069f                        dist[u2] = dist[u1] + 1;
516b                        Q[sizeQ++] = u2;
5af6                 }
b2aa          }
55e8   }
42d2 }
```

```
5b8e inline bool dfs(int u1) {
050a   vis[u1] = true;
e70c   for(int e = last[u1]; e >= 0; e = previous[e]){
176b          int v = head[e];
02f8          int u2 = matching[v];
8af5          if(u2 < 0 || !vis[u2] && dist[u2] == dist[u1] + 1 &&
    dfs(u2)){
59f7                 matching[v] = u1;
adc5                 used[u1] = true;
9f32                 return true;
9484          }
24d9   }
c9bb   return false;
de30 }

aef8 inline int maxMatching() {
ff22   fill(used, used + n1, false);
0d7d   fill(matching, matching + n2, -1);
6531   for(int res = 0; ;){
871f          bfs();
3379          fill(vis, vis + n1, false);
df7a          int f = 0;
11cf          for(int u = 0; u < n1; ++u)
a9e3                 if(!used[u] && dfs(u))
0030                        ++f;
a4cf          if(!f) return res;
c8f8          res += f;
c63a   }
d64c }

     // End of Hopcroft Karp Template

782b /*
a8b4   We will use Dilworths' Theorem and Min-Path-Cover on a DAG to
    solve this problem.
d374   Add an edge from number x to number y (x != y), if y % x == 0.
b34c   Now we've built a dag, and we want to find the size of maximum
    antichain in this DAG.
6ece   We need to find a subset of nodes such that no node in the subset
    can be reached from
9d10   any other node in the subset.

d7bb   By Dilworth's Theorem, Size of maximum antichain = Min Path Cover
    in the DAG.
```

```
052f  Let the size be S.
e9d8  To find the lexicographically smallest anti-chain, fix the
  smallest element and find
15ac  the maximal antichain on the remaining graph. If you can get an
  antichain of size S - 1,
41ee  then the smallest element can be taken. Repeat this process!
2bbe */


0ed2 int main(){
cad8  scanf("%d", &t);
16a5  for(int qq = 1; qq <= t; qq++){
c2bb      scanf("%d", &n);
8fec      values.clear();
ac53      for(int i = 0; i < n; i++){
d59a          scanf("%d", arr + i);
ecbb          values.push_back(arr[i]);
df18      }
e74b      sort(values.begin(), values.end());
7e42      values.resize(unique(values.begin(), values.end()) -
  values.begin());
d00e      n = (int) values.size();
5f72      init(n, n);
415a      for(int i = 0; i < n; i++){
f388          for(int j = 0; j < n; j++){
8e94              if(i == j) continue;
7762              if((values[j] % values[i]) == 0) addEdge(i,
  j);
13c6          }
30e3      }
df30      int max_antichain = n - maxMatching(), ans = max_antichain;
15f5      vector < int > in_sol;
c221      set < int > result;
cb81      for(int i = 0; i < n; i++) result.insert(values[i]);
2df7      while(!result.empty()){
5471          set < int > :: iterator it = result.begin();
1a5f          int check_val = *it;
5982          vector < int > tmp; n = 0;
742c          while((++it) != result.end()){
f8cb              if(*it % check_val){
c522                  ++n;
4fb0                  tmp.push_back(*it);
8200              }
046d          }
aa6a          init(n, n);
ce0b          for(int i = 0; i < n; i++){
eb7d              for(int j = 0; j < n; j++){
2653                  if(i == j) continue;
f044                  if(tmp[j] % tmp[i] == 0) addEdge(i,
  j);
27ff              }
f942          }
5dbd          if(n - maxMatching() == max_antichain - 1){
9285              max_antichain = max_antichain - 1;
2431              in_sol.push_back(check_val);
470a              set < int > :: iterator it2 = result.begin();
299b              while((++it2) != result.end()){
9175                  if(*it2 % check_val == 0){
050e                      set < int > :: iterator it3 =
  it2;
3223                      it3++;
1afa                      result.erase(it2);
8309                      it2 = (--it3);
4c65                  }
2a1e              }
f12d          }
ad63          result.erase(result.begin());
1d16      }
6871      printf("Case %d:", qq);
aceb      for(int i = 0; i < ans; i++) printf(" %d", in_sol[i]);
53d0      printf("\n");
82e3  }
1c6a }
```

3.3 Dinics-hashed

```
     // LIGHTOJ
f425 #include "bits/stdc++.h"
3d1e using namespace std;

     // Dinic's Maxflow Template
3254 const int INF = 1000000000;

4b4a struct Edge {
4bfc  int from, to, cap, flow, index;
b763  Edge(int from, int to, int cap, int flow, int index) :
a5fd  from(from), to(to), cap(cap), flow(flow), index(index) {}
```

```
9751 };

41b3 struct Dinic{
2443  int N;
246e  vector < vector < Edge > > G;
5932  vector < Edge * > dad;
5e68  vector < int > Q;

d3bc  Dinic(int N) : N(N), G(N), dad(N), Q(N) {}

3928  void AddEdge(int from, int to, int cap){
7bd7      G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
3593      if (from == to) G[from].back().index++;
dcd2      G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
969b  }

5e27  long long BlockingFlow(int s, int t){
cbc6      fill(dad.begin(), dad.end(), (Edge *) NULL);
38b8      dad[s] = &G[0][0] - 1;

86cc      int head = 0, tail = 0;
7f83      Q[tail++] = s;

6470      while(head < tail){
574d          int x = Q[head++];
b92b          for (int i = 0; i < G[x].size(); i++){
bcbb              Edge &e = G[x][i];
786a              if(!dad[e.to] && e.cap - e.flow > 0){
3fd4                  dad[e.to] = &G[x][i];
d902                  Q[tail++] = e.to;
16b5              }
a8c8          }
453b      }
9f7a      if (!dad[t]) return 0;

4fd6      long long totflow = 0;
e549      for (int i = 0; i < G[t].size(); i++){
4306          Edge *start = &G[G[t][i].to][G[t][i].index];
d37c          int amt = INF;
2a41          for (Edge *e = start; amt && e != dad[s]; e = dad[e->from]){
4459              if (!e) { amt = 0; break; }
4a0d              amt = min(amt, e->cap - e->flow);
6a2d          }
23a9          if (amt == 0) continue;
da94          for (Edge *e = start; amt && e != dad[s]; e = dad[e->from]) {
41cd              e->flow += amt;
5a92              G[e->to][e->index].flow -= amt;
92a9          }
e0d1          totflow += amt;
8f7b      }
554d      return totflow;
6ad7  }

663d  long long GetMaxFlow(int s, int t){
c08f      long long totflow = 0;
cbb7      while (long long flow = BlockingFlow(s, t)) totflow += flow;
645e      return totflow;
f35f  }
3fc4 };

     // End of Dinic's Maxflow

782b /*
3e78  The min-cut of G(V, E) finds the minimum cost subset E' of E such that in G(V, E \ E'),
2395  Source S and Sink T are not connected! This is exactly what we want in this problem.
dc00  However, we can also remove a vertex instead of an edge.
3fee  Hence, we will split each vertex into 2 nodes and add an edge with weight equal to
dc88  the cost of removing that vertex.

1911  In this graph, min cut will give us the answer. And since we know that min cut = max flow,
6ec5  the problem becomes easy to solve!
e175 */


cd9b int test, m, w;

0ed2 int main(){
eb0b  freopen("ioi.in", "r", stdin);
a454  ios :: sync_with_stdio(false);
60cd  cin >> test;
49b1  for(int qq = 1; qq <= test; qq++){
8043      cin >> m >> w;
690e      Dinic mf(m + m + 2);
0940         int source = 1, sink = m;
e435         for(int i = 2; i < m; i++){
5151                 int cost; cin >> cost;
1dcf                 mf.AddEdge(i, i + m, cost);
7893         }
d85e         for(int i = 1; i <= w; i++){
837d                 int u, v, c;
6250                 cin >> u >> v >> c;
7594                 int nu = u, nv = v;
d3df                 if(u >= 2 and u <= m - 1) nu = u + m;
d4dc                 mf.AddEdge(nu, v, c);
2af9                 if(v >= 2 and v <= m - 1) nv = v + m;
169d                 mf.AddEdge(nv, u, c);
e8c9         }
fbd0         cout << "Case " << qq << ": " << mf.GetMaxFlow(source,
    sink) << '\n';
87a3   }
1c40 }
```

3.4 MinCostMaxFlow-hashed

```
     // LIGHTOJ
f425 #include "bits/stdc++.h"
3d1e using namespace std;


     // Min cost Max flow template

3e39 struct MinimumCostMaximumFlow {

208e     typedef long long Flow;
8d80     typedef long long Cost;
ae03     static const Cost infiniteDistance = 1e18;
338b     static const Cost EPS = 1e-7;
22a2     static const Flow infiniteFlow = 1e18;

4b4a     struct Edge{
8418         int u, v;
8d4f         Flow f, c;
4166         Cost w;
fec6         Edge(int u, int v, Flow f, Flow c, Cost w) : u(u), v(v),
    f(f), c(c), w(w) {}
5987     };
```

```
5569     vector < Edge > e;
e2e5     vector < vector < int > > g;
30fd     int n, source, sink, *prev;
a9d5     Cost *dist;

980c     MinimumCostMaximumFlow(int n) : n(n){
d20d         dist = (Cost*)malloc(sizeof(Cost)*n);
2258         prev = (int*) malloc(sizeof(int)*n);
525f         g.resize(n);
faef     }

33a6     ~MinimumCostMaximumFlow(){
fd98         free(dist);
3aa9         free(prev);
6276         g.clear();
b36e     }

3220     inline void add(int u, int v, Flow c, Cost w){
6cd2         g[u].push_back(e.size());
0290         e.push_back(Edge(u, v, 0, c, w));
             // For residual graph
fddf         g[v].push_back(e.size());
47ae         e.push_back(Edge(v, u, 0, 0, -w));
7240     }

22ce     inline pair < Cost, Flow > getMaxFlow(int source, int sink){
3189         this -> source = source;
04d7         this -> sink = sink;
2d5d         for(int i = 0; i < (int) e.size(); i++) e[i].f = 0;
fe2f         Flow flow = 0;
e603         Cost cost = 0;
0f3a         while(bellmanFord()){
02b1             int u = sink;
49f1             Flow pushed = infiniteFlow;
a48f             Cost pushCost = 0;
d4e8             while(u != source){
1533                 int id = prev[u];
f4d2                 pushed = min(pushed, e[id].c - e[id].f);
20da                 pushCost += e[id].w;
c07a                 u = e[id].u;
d67e             }
b6f6             u = sink;
1c7b             while(u != source){
5a11                 int id = prev[u];
3c17                e[id].f += pushed;
355a                e[id ^ 1].f -= pushed;
c51a                u = e[id].u;
d655            }
4f01            flow += pushed;
a788            cost += pushCost * pushed;
4541        }
cac9        return make_pair(cost, flow);
4e2b    }

1b90    inline bool bellmanFord(){
66a8        for(int i = 0; i < n; ++i) dist[i] = infiniteDistance;
c907        dist[source] = 0;
2a19        for(int k = 0; k < n; ++k){
5f7a            bool update = false;
58b6            for(int id = 0; id < (int) e.size(); ++id){
be9a                int u = e[id].u;
d594                int v = e[id].v;
33ed                if(dist[u] + EPS >= infiniteDistance) continue;
8c5f                Cost w = e[id].w;
be4e                if(e[id].f < e[id].c && dist[v] > dist[u] + w + EPS){
d4dd                    dist[v] = dist[u] + w;
cd40                    prev[v] = id;
b6b1                    update = true;
8dc8                }
4413            }
d7c5            if(!update) break;
2ec3        }
a262        return (dist[sink] + EPS) < (infiniteDistance);
156e    }

        // After running mcmf, e[id].f has the flow which has passed
            through that edge in the optimal soln
180f    inline void displayEdges(){
f5b0        cout << "******" << '\n';
f160        for(int i = 0; i < (int) e.size(); ++i)
ac24            cout << e[i].u << " " << e[i].v << " " << e[i].f << " "
    << e[i].c << " " << e[i].w <<"\n";
e06a        cout << "******" << '\n';
577e    }
37ad };
```

```
7ae9 const int N = 1e2 + 2;

313b int t, n, m;
0ae5 int a[N][N], in[N][N], out[N][N];

f53a inline bool is_valid(int x, int y){
5e6c   return (x >= 1 and x <= n and y >= 1 and y <= m);
628e }

0ed2 int main(){
eb0b   freopen("ioi.in", "r", stdin);
07af   cin >> t;
6168   for(int qq = 1; qq <= t; qq++){
2a75          cin >> n >> m;
6d15          for(int i = 1; i <= n; i++)
6bd5                 for(int j = 1; j <= m; j++)
e9db                        cin >> a[i][j];
3a2e          int cur_time = 0;
869b          for(int i = 1; i <= n; i++){
6c50                 for(int j = 1; j <= m; j++){
8201                        in[i][j] = ++cur_time;
7e16                        out[i][j] = in[i][j] + (n * m);
b38d                 }
6de1          }
61f2          MinimumCostMaximumFlow mcmf(2 * n * m + 1);
53be          int source = out[1][1], sink = in[n][m];
4eaf          for(int i = 1; i <= n; i++){
7634                 for(int j = 1; j <= m; j++){
bf61                        mcmf.add(in[i][j], out[i][j], 1, -a[i][j]);
440c                        if(is_valid(i, j + 1))
92c2                               mcmf.add(out[i][j], in[i][j + 1], 1,
    0);
ba01                        if(is_valid(i + 1, j))
1370                               mcmf.add(out[i][j], in[i + 1][j], 1,
    0);
80e6                 }
347a          }
41c8          cout << "Case " << qq << ": " << (a[1][1] + a[n][m]
    -mcmf.getMaxFlow(source, sink).first);
931d          cout << "\n";
ece5   }
2f1a }
```

4 Graphs and Trees

4.1 Auxiliary Tree

4.1.1 AuxiliaryTree-hashed

```
f425 #include "bits/stdc++.h"
3d1e using namespace std;

42e7 const int N  = 1e5 + 5;
a6af const int LN = 18;
a64b const int INF = 1e8 + 8;

8ed5 int n, q, cur_time, len, ans;
cf29 int tin[N], tout[N], depth[N], parent[N], val[N], dp[LN][N];
5fba bool important[N];
5c42 vector < int > adj[N], aux[N];
dbf8 vector < int > nodes;

4a6d inline void dfs_prep(int u, int p){
faaa   tin[u] = ++cur_time;
2a88   dp[0][u] = parent[u] = p;
53a3   for(int i = 1; i < LN; i++) dp[i][u] = dp[i - 1][dp[i - 1][u]];
c77a   for(int v : adj[u]){
ed78           if(v != p){
049a                   depth[v] = depth[u] + 1;
b93e                   dfs_prep(v, u);
f5b4           }
a7d0   }
6c35   tout[u] = cur_time;
ab1c }

a8a2 inline int lca(int u, int v){
00e6   if(depth[u] < depth[v]) swap(u, v);
e1bb   for(int i = LN - 1; i >= 0; i--){
0fa1           if(depth[u] - (1 << i) >= depth[v])
77c2                   u = dp[i][u];
13c3   }
09d8   if(u == v) return u;
dfb2   for(int i = LN - 1; i >= 0; i--){
c203           if(dp[i][u] != dp[i][v])
6979                   u = dp[i][u], v = dp[i][v];
cb36   }
93d4   return parent[u];
a4e3 }

7b9f inline bool compare(int u, int v){
c8a7   return (tin[u] < tin[v]);
3e38 }

9697 inline void clean(vector < int > &x){
7263   sort(x.begin(), x.end(), compare);
81db   x.resize(unique(x.begin(), x.end()) - x.begin());
cc26   len = (int) nodes.size();
361c }

b2f1 inline void dfs(int u){
b007   val[u] = INF;
eedc   int min_val = INF, noob_child = 0;
33d5   for(int v : aux[u]){
f107           dfs(v);
5bb5           if(val[v] != INF) ++noob_child;
0700           min_val = min(min_val, val[v]);
0045   }
fd2b   if(!important[u]){
6678           if(noob_child > 1) ans += 1;
305c           else val[u] = min_val;
e1ff   }
9b96   else{
714f           val[u] = 1;
1a98           ans += noob_child;
c0a9   }
4e78 }

4922 inline bool is_ancestor(int u, int v){
a77f   return ((tin[v] >= tin[u]) && (tin[v] <= tout[u]));
fd46 }

1696 inline void solve(bool rekt){
21b6   if(rekt){
df12           printf("-1\n");
44ff           return;
fa5a   }
ac07   clean(nodes);
ff93   for(int i = 0; i < len - 1; i++){
6d95           int lc = lca(nodes[i], nodes[i + 1]);
9e69           nodes.push_back(lc);
4c8e   }
f76d   clean(nodes);
283a   stack < int > ancestors;
bc40   int root = nodes[0];
0547   ancestors.push(root);
8838   for(int i = 1; i < len; i++){
d36e          while(!is_ancestor(ancestors.top(), nodes[i]))
9c21                 ancestors.pop();
5065          int p = ancestors.top();
33bc          aux[p].push_back(nodes[i]);
1dbb          ancestors.push(nodes[i]);
d890   }
d63a   ans = 0; dfs(root);
36e1   printf("%d\n", ans);
ad37   for(int node : nodes) aux[node].clear();
bd14 }

0ed2 int main(){
eb14   cin >> n;
6a1c   for(int i = 1; i < n; i++){
4032          int u, v;
ca22          cin >> u >> v;
fb36          adj[u].push_back(v);
b887          adj[v].push_back(u);
3db9   }
54de   dfs_prep(1, 1);
9855   cin >> q;
c793   while(q--){
1c01          int k;
a14e          cin >> k;
36c2          for(int i = 1; i <= k; i++){
0863                 int node;
a615                 cin >> node;
e888                 important[node] = true;
1031                 nodes.push_back(node);
88fc          }
b6bf          bool rekt = false;
b28a          for(int node : nodes){
9bc6                 if((parent[node] != node) &&
    (important[parent[node]]))
caa7                        rekt = true;
3e28          }
2dbe          solve(rekt);
6ada          for(int node : nodes) important[node] = false;
cb09          nodes.clear();
4e25   }
2a0c }
```

4.2 Block Cut Tree

4.2.1 BlockCutTree-hashed

```
     // Codeforces - Tourists (Some Div 1E)
8c39 #include <bits/stdc++.h>
3a00 using namespace std;

f22f const int INF = 1e9 + 333;
44e8 const int MAX = (1 << 18);
9ab7 const int LN = 18;

0084 int n, m, q, timekeeper, cnt_bcc, sz, w[MAX];
3525 int disc[MAX], low[MAX], cut[MAX], bcc[MAX], baap[MAX], is_cut[MAX];
7223 int tree[MAX << 1], timer, chainPos[MAX], head[MAX], root;
09ae int sub[MAX], depth[MAX], dp[LN][MAX], vis[MAX];
ddd0 vector < pair < int, int > > adj[MAX];
70a5 vector < int > bc_tree[MAX], in_bcc[MAX];
dcd2 pair < int, int > temp[MAX];
db0c multiset < int > costs[MAX];

5ac7 inline void make_tree(int p_id = -1, int p = 0, int x = 1) {
abbc     vis[x] = 1;
1e7b     temp[sz++] = make_pair(x, p);
77e1     low[x] = disc[x] = ++timekeeper;
d4a1     for(auto it : adj[x]) {
5edf         int u = it.first;
7df3         int e = it.second;
5033         if(e != p_id) {
6693             if(!vis[u]) {
723a                 make_tree(e, x, u);
51ca                 low[x] = min(low[x], low[u]);
0ad6                 if(low[u] >= disc[x]) {
be29                     cut[x] = 1;
d0da                     cnt_bcc++;
d491                     while(temp[sz] != make_pair(u, x)) {
76ea                         bcc[temp[sz - 1].first] = cnt_bcc;
2374                         bcc[temp[sz - 1].second] = cnt_bcc;
e998                         sz--;
c731                     }
2d5b                     in_bcc[x].push_back(cnt_bcc);
0c85                     baap[cnt_bcc] = x;
2819                 }
c93d             }
5f3f             else
e1ad                 low[x] = min(low[x], disc[u]);
6f70         }
8306     }
3465 }

86bf inline void pre(int p = 0, int x = root) {
cd3d     sub[x] = 1;
0034     dp[0][x] = p;
9a49     depth[x] = depth[p] + 1;
5e0b     for(int i = 1; i < LN; i++) dp[i][x] = dp[i - 1][dp[i - 1][x]];
89f7     for(auto it : bc_tree[x]) {
0639         int u = it;
e6e6         if(u != p) {
c788             pre(x, u);
6a77             sub[x] += sub[u];
bb2e         }
75a4     }
23d0 }

2217 inline void hld(int p = -1, int x = root, int h = root) {
6d1c     head[x] = h;
d440     chainPos[x] = ++timer;
63a7     int rdcount = -1, rajat = -1; // pro-child
0e42     for(auto it : bc_tree[x]) {
b3be         int u = it;
d594         if(u != p and sub[u] > rdcount) {
4e53             rdcount = u;
9052             rajat = u;
94ff         }
fcda     }
598f     if(rajat != -1) hld(x, rajat, h);
0907     for(auto it : bc_tree[x]) {
f6b9         int u = it;
4fd5         if(u != p and u != rajat)
df25             hld(x, u, u);
2e84     }
2109 }

ecaf inline void update(int x, int k) {
3737     tree[x += MAX] = k;
95eb     while(x > 1) {
0cab         x >>= 1;
4b00         tree[x] = min(tree[x + x], tree[x + x + 1]);
0225     }
286c }
```

```
ca8c inline int query(int l, int r) {
a5cd     int res = INF;
026c     for(l += MAX, r += MAX; l <= r; l >>= 1, r >>= 1) {
2aab         if(l & 1) res = min(res, tree[l++]);
ac2a         if(~r & 1) res = min(res, tree[r--]);
551c     }
aa97     return res;
bd29 }

c47a inline int get_lca(int x, int y) {
4e5b     if(depth[x] < depth[y]) swap(x, y);
91cb     for(int i = LN - 1; i >= 0; i--)
3443         if(depth[x] - (1 << i) >= depth[y])
f0f6             x = dp[i][x];
8245     if(x == y) return x;
4239     for(int i = LN - 1; i >= 0; i--) {
edc2         if(dp[i][x] != dp[i][y]) {
f345             x = dp[i][x];
bd38             y = dp[i][y];
c594         }
a651     }
b7c3     return dp[0][x];
1dc3 }

105c inline int qmin(int x, int up) {
0478     int res = INF;
f106     while(depth[x] >= depth[up]) {
d052         res = min(res, query(max(chainPos[up], chainPos[head[x]]),
    chainPos[x]));
9218         x = dp[0][head[x]];
c4ed     }
ceeb     return res;
5e0a }

0ed2 int main () {

7eda     scanf("%d %d %d", &n, &m, &q);

9186     for(int i = 1; i <= n; i++) {
c982         scanf("%d", w + i);
1631     }

918a     for(int i = 1; i <= m; i++) {
e75a         int x, y;
933f            scanf("%d %d", &x, &y);
980a            adj[x].push_back(make_pair(y, i));
30c3            adj[y].push_back(make_pair(x, i));
19fb        }

366d        make_tree();

2d76        for(int i = n; i >= 1; i--) {
3345            if(cut[i]) {
0a3f                cut[i] = ++cnt_bcc;
6186                is_cut[cnt_bcc] = i;
3371            }
89e6        }

8c16        root = cnt_bcc;

9186        for(int i = 1; i <= n; i++) {
d23c            if(cut[i]) {
9328                bc_tree[bcc[i]].push_back(cut[i]);
0bea                bc_tree[cut[i]].push_back(bcc[i]);
2c4e                for(auto it : in_bcc[i]) {
6f9c                    int x = it;
a748                    bc_tree[cut[i]].push_back(x);
103b                    bc_tree[x].push_back(cut[i]);
d8fc                }
e6ba            }
d748        }

9186        for(int i = 1; i <= n; i++) {
78a2            sort(bc_tree[i].begin(), bc_tree[i].end());
d974            bc_tree[i].resize(unique(bc_tree[i].begin(),
    bc_tree[i].end()) - bc_tree[i].begin());
a6b6        }

5af7        pre();
dd29        hld();

9186        for(int i = 1; i <= n; i++) {
b1fb            costs[bcc[i]].insert(w[i]);
ddf2        }

696e        for(int i = 1; i <= cnt_bcc; i++) {
1266            update(chainPos[i], costs[i].size() ? *costs[i].begin() :
    INF);
30ee        }
```

```
6ca3        while(q--){
0771            char c;
9585            int x, y;
d771            scanf(" %c %d %d", &c, &x, &y);
c7c6            if(c == 'A') {
2dd9                if(x == y) {
e027                    printf("%d\n", w[x]);
1c28                    continue;
409c                }
d274                x = cut[x] ? cut[x] : bcc[x];
c064                y = cut[y] ? cut[y] : bcc[y];
fe7c                int lca = get_lca(x, y);
1786                int res = min(qmin(x, lca), qmin(y, lca));
6810                if(is_cut[lca]) res = min(res, w[is_cut[lca]]);
3cee                else if(baap[lca]) res = min(res, w[baap[lca]]);
d35f                printf("%d\n", res);
fee7            }
1799            else {
920c                costs[bcc[x]].erase(costs[bcc[x]].find(w[x]));
2b7b                costs[bcc[x]].insert(w[x] = y);
cf78                update(chainPos[bcc[x]], *costs[bcc[x]].begin());
c606            }
364d        }
69cf    }
```

4.3 Bridge Tree

4.3.1 BridgeTree-hashed

```
     // Can you add one edge to a connected graph (with multiedges), to
         remove all bridges in it?
f425 #include "bits/stdc++.h"
3d1e using namespace std;

42e7 const int N = 1e5 + 5;

62cd int n, m, cur_time, component_id;
a096 int a[N * 2], b[N * 2], is_bridge[N * 2];
b13c int disc[N], low[N], component[N];
d30c vector < int > adj[N], tree[N];

c261 inline void find_bridges(int u, int p){
```

```
ab99  disc[u] = low[u] = ++cur_time;
2df5  for(int i = 0; i < (int) adj[u].size(); i++){
f759      int edge_id = adj[u][i];
2bd1      int v = a[edge_id] ^ b[edge_id] ^ u;
7c30      if(edge_id == p) continue;
6e18      if(!disc[v]){
9797          find_bridges(v, edge_id);
6e24          low[u] = min(low[u], low[v]);
c352          if(low[v] > disc[u]) is_bridge[edge_id] = true;
9667      }
10d4      else low[u] = min(low[u], disc[v]);
a0fb  }
dd7a }

844b inline void explore(int u){
c21f  component[u] = component_id;
eec1  for(int i = 0; i < (int) adj[u].size(); i++){
6d38      int edge_id = adj[u][i];
e6e1      int v = a[edge_id] ^ b[edge_id] ^ u;
20a7      if(component[v] || is_bridge[edge_id]) continue;
c0c5      explore(v);
2e7b  }
d90e }

0ed2 int main(){
b19b  cin >> n >> m;
5c52  for(int i = 1; i <= m; i++){
cd1e      cin >> a[i] >> b[i];
1672      adj[a[i]].push_back(i);
8ddf      adj[b[i]].push_back(i);
fc13  }
0ec0  find_bridges(1, 0);
f181  for(int i = 1; i <= n; i++){
ece5      if(!component[i]){
c85b          ++component_id;
3ed4          explore(i);
a18b      }
5d71  }
2924  for(int i = 1; i <= m; i++){
0291      if(is_bridge[i]){
9d33          int u = component[a[i]], v = component[b[i]];
763b          tree[u].push_back(v);
f2ee          tree[v].push_back(u);
77ea      }
53c2  }
171e  int cnt_leaves = 0;
14f6  for(int i = 1; i <= component_id; i++){
1e35      cnt_leaves += (((int) tree[i].size()) == 1);
a88c  }
2b2f  if(cnt_leaves <= 2) cout << "YES\n";
c38b  else cout << "NO\n";
5e61 }
```

4.4 Centroid Decomposition

4.4.1 CentroidDecomposition-hashed

```
782b /*
1572  Given a Tree T and Q queries, each of the form (v, l) :
46d5  Each query returns number of vertices u such that distance(v, u)
    <= l

ec60  Centroid Decomposition!
4815 */


8c39 #include <bits/stdc++.h>
3a00 using namespace std;

d699 const int MAX = 100005;
1c3e const int LN = 20;

f547 int n, q;
bc84 vector < pair < int, long long > > adj[MAX];
a61f int done[MAX], parent[MAX], depth[MAX], sub[MAX];
daec long long dist[LN][MAX];
7e86 vector < long long > val[MAX], valp[MAX];

b9a8 void dfs(int u, int p){
e584  sub[u] = 1;
9297  for(auto v : adj[u]){
9f06      if(v.first == p || done[v.first]) continue;
cddc      dfs(v.first, u);
00f7      sub[u] += sub[v.first];
b87a  }
d5be }

5aed int find(int u, int p, int tar){
```

```
48f8   for(auto v : adj[u]){
f6b9          if(v.first == p || done[v.first]) continue;
7ab1          if(sub[v.first] > tar) return find(v.first, u, tar);
8ba8   }
304a   return u;
51ff }

0ece void explore(int u, int p, long long d, int cur){
843e   val[cur].push_back(d);
49ed   dist[depth[cur]][u] = d;
88fc   for(auto v : adj[u]){
6b2a          if(done[v.first] || v.first == p) continue;
8cb4          explore(v.first, u, d + v.second, cur);
a418   }
c55d }

f663 void decompose(int u, int p){
06ba   dfs(u, p);
b614   int centroid = find(u, p, sub[u] / 2);
82ec   parent[centroid] = p;
4af4   done[centroid] = true;
fa59   depth[centroid] = (p == 0) ? (0) : (depth[p] + 1);
f0dc   explore(centroid, p, 0, centroid);
060c   sort(val[centroid].begin(), val[centroid].end());
316d   for(auto v : adj[centroid]){
42a7          if(done[v.first]) continue;
aa30          decompose(v.first, centroid);
852c   }
6454 }

daa6 void preprocess(){
c2eb   for(int i = 1; i <= n; i++){
9998          int cur = i;
04f4          while(parent[cur] != 0){
74f5                 valp[cur].push_back(dist[depth[parent[cur]]][i]);
41fd                 cur = parent[cur];
ea72          }
972e   }
aa71   for(int i = 1; i <= n; i++) sort(valp[i].begin(), valp[i].end());
8d2e }

f6ad int query(int v, long long l){
76f6   int ans = upper_bound(val[v].begin(), val[v].end(), l) -
    val[v].begin();
c175   int cur = v;
6741   while(parent[cur] != 0){
8859          long long d = dist[depth[parent[cur]]][v];
5f34          int tot = upper_bound(val[parent[cur]].begin(),
    val[parent[cur]].end(), l - d) - val[parent[cur]].begin();
70d0          int ext = upper_bound(valp[cur].begin(), valp[cur].end(),
    l - d) - valp[cur].begin();
cb7e          ans += tot - ext;
fe76          cur = parent[cur];
b78e   }
10d5   return ans;
a8fb }

0ed2 int main(){

7a03   scanf("%d %d\n", &n, &q);
130d   for(int i = 1; i < n; i++){
6f10          int u, v;
0917          long long l;
8062          scanf("%d %d %lld\n", &u, &v, &l);
77ca          adj[u].push_back(make_pair(v, l));
31c7          adj[v].push_back(make_pair(u, l));
39f3   }

c563   decompose(1, 0);
91f2   preprocess();
8980   while(q--){
fda6          int v;
adba          long long l;
4eb9          scanf("%d %lld\n", &v, &l);
8242          printf("%d\n", query(v, l));
146f   }
78de }
```

4.5 Euler Path

4.5.1 EulerPath-hashed

```
4b0a struct Edge;
3804 typedef list<Edge>::iterator iter;

6629 struct Edge
4b4a {
```

```
0b22  int next_vertex;
99b1  iter reverse_edge;

4695  Edge(int next_vertex)
df49      :next_vertex(next_vertex)
0a09      { }
6a7a };

c1d8 const int max_vertices = ;
26bc int num_vertices;
40d3 list<Edge> adj[max_vertices];     // adjacency list

a86c vector<int> path;

b576 void find_path(int v)
b5d0 {
0463  while(adj[v].size() > 0)
1858  {
6e90      int vn = adj[v].front().next_vertex;
4ba9      adj[vn].erase(adj[v].front().reverse_edge);
d6df      adj[v].pop_front();
1191      find_path(vn);
88f1  }
f62e  path.push_back(v);
77cc }

aa2a void add_edge(int a, int b)
552a {
2b8d  adj[a].push_front(Edge(b));
8b2e  iter ita = adj[a].begin();
223b  adj[b].push_front(Edge(a));
8a1b  iter itb = adj[b].begin();
5bb3  ita->reverse_edge = itb;
0516  itb->reverse_edge = ita;
b055 }
```

4.6 Heavy Light Decomposition

4.6.1 HLD-hashed

```
8c39 #include <bits/stdc++.h>
96f2 #define rf freopen("inp.in", "r", stdin)
88c6 using namespace std;

fc54 const int MAX = 10005;

362c int t, n, x, y, a[MAX], b[MAX], c[MAX];
156c int depth[MAX], heavy[MAX], root[MAX], parent[MAX], sub[MAX];
8090 int edgeToNode[MAX], nodeToEdge[MAX], pos[MAX], tree[MAX << 2];
27f4 vector < pair < int, int > > adj[MAX];
77bb char str[MAX];

c559 inline void dfs(int u, int p){
020f  sub[u] = 1;
4b3b  int mx = 0;
8646  for(int i = 0; i < adj[u].size(); i++){
f965      int v = adj[u][i].first;
e6ca      if(v == p) continue;
ef0b      edgeToNode[adj[u][i].second] = v;
5034      nodeToEdge[v] = adj[u][i].second;
ee55      parent[v] = u;
d4a8      depth[v] = depth[u] + 1;
21a3      dfs(v, u);
efc8      sub[u] += sub[v];
adaf      if(sub[v] > mx){
de41          mx = sub[v];
e221          heavy[u] = v;
0f6c      }
6006  }
337d }

b276 inline void update(int node, int l, int r, int idx, int val){
889a  if(l == r){
a4d0      tree[node] = val;
0b87      return;
3821  }
de91  int mid = l + r >> 1;
1ab0  if(mid >= idx) update(node + node, l, mid, idx, val);
eb6a  else update(node + node + 1, mid + 1, r, idx, val);
cab5  tree[node] = max(tree[node + node], tree[node + node + 1]);
ae28 }

586e inline int query(int node, int l , int r, int qs, int qe){
cbe0  if(l > qe or r < qs)  return 0;
217a  if(l >= qs and r <= qe) return tree[node];
b2f4  int mid = l + r >> 1;
289c  return max( query(node + node, l, mid, qs, qe), query(node + node
  + 1, mid + 1, r, qs, qe) );
e139 }

d407 inline void hld(){
459f   dfs(1, 0);
092c   for(int i = 1, curPos = 0; i <= n; i++){
12af          if(parent[i] == -1 || heavy[parent[i]] != i){
60de                 for(int j = i; j != -1; j = heavy[j])
ae08                        root[j] = i, pos[j] = ++curPos;
450d          }
6a55   }
b8a1   for(int i = 2; i <= n; i++) update(1, 1, n, pos[i],
    c[nodeToEdge[i]]);
0db8 }

eb8f inline int query(int u, int v){
cbd2   int mx = 0;
bf77   for(; root[u] != root[v]; v = parent[root[v]]){
988d          if(depth[root[u]] > depth[root[v]]) swap(u, v);
e071          mx = max(mx, query(1, 1, n, pos[root[v]], pos[v]));
8f7e   }
fb66   if(depth[u] > depth[v]) swap(u, v);
1367   mx = max(mx, query(1, 1, n, pos[u] + 1, pos[v]));
89c7   return mx;
3c33 }

d9dd inline void solve(){
2da2   scanf("%d", &n);
67e0   memset(tree, 0, sizeof tree);
61b5   for(int i = 1; i <= n; i++){
6f29          adj[i].clear();
3f77          heavy[i] = parent[i] = edgeToNode[i] = nodeToEdge[i] = -1;
5513          depth[i] = sub[i] = 0;
9ad5   }
7d00   for(int i = 1; i < n; i++){
ff74          scanf("%d %d %d\n", &a[i], &b[i], &c[i]);
9b3d          adj[a[i]].push_back(make_pair(b[i], i));
b474          adj[b[i]].push_back(make_pair(a[i], i));
a5de   }
218e   hld();
72cb   while(true){
8cb3          scanf("%s", str);
f2d0          if(str[0] == ’D’) break;
cb01          else if(str[0] == ’C’){
5f04                 scanf("%d %d\n", &x, &y);
6989                 update(1, 1, n, pos[edgeToNode[x]], y);
4b31          }
fcc3          else{
800c                 scanf("%d %d\n", &x, &y);
f6c5                 printf("%d\n", query(x, y));
2fcb          }
5903   }
1ab5 }

0ed2 int main(){
311c   rf;
331f   scanf("%d", &t);
b512   while(t--) solve();
95d5 }
```

4.7 Mo’s On Trees

4.7.1 Mo’sOnTrees

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

const int MAXN = 40005;
const int MAXM = 100005;
const int LN = 19;

int N, M, K, cur, A[MAXN], LVL[MAXN], DP[LN][MAXN];
int BL[MAXN << 1], ID[MAXN << 1], VAL[MAXN], ANS[MAXM];
int d[MAXN], l[MAXN], r[MAXN];
bool VIS[MAXN];
vector < int > adjList[MAXN];

struct query{
       int id, l, r, lc;
       bool operator < (const query& rhs){
              return (BL[l] == BL[rhs.l]) ? (r < rhs.r) : (BL[l] <
                  BL[rhs.l]);
       }
}Q[MAXM];

// Set up Stuff
void dfs(int u, int par){
       l[u] = ++cur;
       ID[cur] = u;
```

```
        for (int i = 1; i < LN; i++) DP[i][u] = DP[i - 1][DP[i - 1][u]];
        for (int i = 0; i < adjList[u].size(); i++){
                int v = adjList[u][i];
                if (v == par) continue;
                LVL[v] = LVL[u] + 1;
                DP[0][v] = u;
                dfs(v, u);
        }
        r[u] = ++cur; ID[cur] = u;
}

// Function returns lca of (u) and (v)
inline int lca(int u, int v){
        if (LVL[u] > LVL[v]) swap(u, v);
        for (int i = LN - 1; i >= 0; i--)
                if (LVL[v] - (1 << i) >= LVL[u]) v = DP[i][v];
        if (u == v) return u;
        for (int i = LN - 1; i >= 0; i--){
                if (DP[i][u] != DP[i][v]){
                        u = DP[i][u];
                        v = DP[i][v];
                }
        }
        return DP[0][u];
}

inline void check(int x, int& res){
        // If (x) occurs twice, then don't consider it's value
        if ( (VIS[x]) and (--VAL[A[x]] == 0) ) res--;
        else if ( (!VIS[x]) and (VAL[A[x]]++ == 0) ) res++;
        VIS[x] ^= 1;
}

void compute(){

        // Perform standard Mo's Algorithm
        int curL = Q[0].l, curR = Q[0].l - 1, res = 0;

        for (int i = 0; i < M; i++){

                while (curL < Q[i].l) check(ID[curL++], res);
                while (curL > Q[i].l) check(ID[--curL], res);
                while (curR < Q[i].r) check(ID[++curR], res);
                while (curR > Q[i].r) check(ID[curR--], res);
```

```
                int u = ID[curL], v = ID[curR];

                // Case 2
                if (Q[i].lc != u and Q[i].lc != v) check(Q[i].lc, res);

                ANS[Q[i].id] = res;

                // Case 2
                if (Q[i].lc != u and Q[i].lc != v) check(Q[i].lc, res);
        }

        for (int i = 0; i < M; i++) printf("%d\n", ANS[i]);
}

int main(){

        int u, v, x;

        while (scanf("%d %d", &N, &M) != EOF){

                // Cleanup
                cur = 0;
                memset(VIS, 0, sizeof(VIS));
                memset(VAL, 0, sizeof(VAL));
                for (int i = 1; i <= N; i++) adjList[i].clear();

                // Inputting Values
                for (int i = 1; i <= N; i++) scanf("%d", &A[i]);
                memcpy(d + 1, A + 1, sizeof(int) * N);

                // Compressing Coordinates
                sort(d + 1, d + N + 1);
                K = unique(d + 1, d + N + 1) - d - 1;
                for (int i = 1; i <= N; i++) A[i] = lower_bound(d + 1, d +
                    K + 1, A[i]) - d;

                // Inputting Tree
                for (int i = 1; i < N; i++){
                        scanf("%d %d", &u, &v);
                        adjList[u].push_back(v);
                        adjList[v].push_back(u);
                }

                // Preprocess
                DP[0][1] = 1;
```

```
              dfs(1, -1);
              int size = sqrt(cur);

              for (int i = 1; i <= cur; i++) BL[i] = (i - 1) / size + 1;

              for (int i = 0; i < M; i++){
                     scanf("%d %d", &u, &v);
                     Q[i].lc = lca(u, v);
                     if (l[u] > l[v]) swap(u, v);
                     if (Q[i].lc == u) Q[i].l = l[u], Q[i].r = l[v];
                     else Q[i].l = r[u], Q[i].r = l[v];
                     Q[i].id = i;
              }

              sort(Q, Q + M);
              compute();
       }
}
```

4.8 Reachability Tree

4.8.1 ReachabilityTree

```
8c39  #include <bits/stdc++.h>
3a00  using namespace std;

43e7  const int N = 3e5 + 5;
ee6b  const int LN = 21;

4b48  struct edge{
bbc9    int u, v, w;
0a66    friend bool operator < (edge x, edge y){
906e           return (x.w < y.w);
74fe    }
83cf  }edges[N];

8e4f  int n, m, q, root;
9f13  vector < int > adj[N];
bc37  int t[N], val[N];
d5b6  int tin[N], distinct[N], deg[N], parent[N];
5236  int timer, head[N];
9925  vector < int > values;
30de  vector < int > nodes[N];
```

```
c195  int depth[N], dp[N][LN];

63c4  inline int find(int x){
42bd    if(parent[x] == x) return x;
e280    return parent[x] = find(parent[x]);
0769  }

0b63  inline int lca(int x, int y){
ad62    if(depth[x] < depth[y]) swap(x, y);
6516    for(int i = LN - 1; i >= 0; i--){
ce2d           if(depth[x] - (1 << i) >= depth[y])
21c2                  x = dp[x][i];
1173    }
63c0    if(x == y) return x;
c7d8    for(int i = LN - 1; i >= 0; i--){
5d07           if(dp[x][i] != dp[y][i]){
74e4                  x = dp[x][i];
f237                  y = dp[y][i];
bfec           }
6582    }
fbbb    return dp[x][0];
dfa0  }

66fd  inline void dfs_init(int u, int p, int rt){
a16f    dp[u][0] = p;
f1d9    tin[u] = ++timer;
06f6    head[u] = rt;
5d33    for(int i = 1; i < LN; i++) dp[u][i] = dp[dp[u][i - 1]][i - 1];
470e    for(int v : adj[u]){
eedc           if(v != p){
44a0                  depth[v] = depth[u] + 1;
9c67                  dfs_init(v, u, rt);
3c9e           }
f199    }
cff1  }

a5e8  inline void dfs_find(int u, int p){
98c8    for(int v : adj[u]){
10ea           if(v != p){
028e                  dfs_find(v, u);
e028                  distinct[u] += distinct[v];
477c           }
e246    }
376f  }
```

```
ff2b inline bool cmp(int x, int y){
9a68   return (tin[x] < tin[y]);
44ae }

5b34 inline void build(){
dd68   sort(edges + 1, edges + 1 + m);
d1f7   for(int i = 1; i <= 3 * n; i++) parent[i] = i;
43e0   root = n;
61ab   for(int i = 1; i <= m; i++){
05ac          int u = find(edges[i].u), v = find(edges[i].v), w =
    edges[i].w;
a92c          if(u == v) continue;
1b28          ++root;
f2a5          parent[u] = parent[v] = root;
5fd0          val[root] = w;
62cc          adj[root].push_back(u);
e5af          adj[root].push_back(v);
e3a9          deg[u]++, deg[v]++;
4f60   }
91c0   for(int i = root; i >= 1; i--){
9a58          if(!deg[i]) dfs_init(i, i, i);
c4af   }
c664   for(int i = 1; i <= n; i++){
9b8c          if(nodes[i].size()){
cc7d                 sort(nodes[i].begin(), nodes[i].end(), cmp);
c7f1                 for(int j = 0; j < (int) nodes[i].size() - 1; j++){
3809                        int x = nodes[i][j], y = nodes[i][j + 1];
d67e                        if(head[x] == head[y]) distinct[lca(x, y)]--;
e8b5                        distinct[x]++;
af38                 }
1e61                 distinct[nodes[i][(int) nodes[i].size() - 1]]++;
088e          }
7039   }
c8ff   for(int i = root; i >= 1; i--){
8dd9          if(!deg[i]) dfs_find(i, i);
cc13   }
9e1d }

2a93 void solve(int u, int v, int k){
ee83   if((head[u] != head[v]) || (distinct[head[u]] < k)){
39ab          printf("-1\n");
7828          return;
43bc   }
d163   int lc = lca(u, v);
fad0   if(distinct[lc] >= k){
b737          printf("%d\n", val[lc]);
e9fb          return;
df32   }
f221   int node = lc;
2649   for(int i = LN - 1; i >= 0; i--){
67dd          if(distinct[dp[node][i]] < k){
a040                 node = dp[node][i];
057f          }
f856   }
eafc   node = dp[node][0];
8ba4   printf("%d\n", val[node]);
2420 }

0ed2 int main(){
084a    scanf("%d %d %d", &n, &m, &q);
b482    for(int i = 1; i <= n; i++){
dc86          scanf("%d", t + i);
1ce1          values.push_back(t[i]);
089a    }
07fe    sort(values.begin(), values.end());
533a    values.resize(unique(values.begin(), values.end()) -
    values.begin());
0caf    for(int i = 1; i <= n; i++){
2f41          t[i] = lower_bound(values.begin(), values.end(), t[i]) -
    values.begin() + 1;
ae00          nodes[t[i]].push_back(i);
050d    }
1708    for(int i = 1; i <= m; i++){
a5f4          scanf("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].w);
a552    }
a509    build();
1528    for(int i = 1; i <= q; i++){
a8cb          int u, v, k;
2887          scanf("%d %d %d", &u, &v, &k);
0eac          solve(u, v, k);
6008    }
437d }
```

4.9 Shortest Paths

4.9.1 CountShortestPaths-hashed

```
     // Undirected, Weighted Graph without self loops and multiple edges.
     // dist[i][j] = Shortest Path from (i) to (j)
     // num[i][j] = Number of Shortest Paths from (i) to (j)

     // NOI Social Network (WciPeg)

f425 #include "bits/stdc++.h"
3d1e using namespace std;

7ae9 const int N = 1e2 + 2;
e243 const long long INF = 1e12;

1c16 int n, m, u, v, w;
8f8d long long dist[N][N], num[N][N];

0ed2 int main(){
eb0b  freopen("ioi.in", "r", stdin);
8dfa  scanf("%d %d", &n, &m);
6cc0  for(int i = 1; i <= n; i++){
41a9      for(int j = 1; j <= n; j++){
e878          dist[i][j] = INF;
c73e      }
3c5c      dist[i][i] = 0;
e19f  }
5e7a  for(int i = 1; i <= m; i++){
9a08      scanf("%d %d %d", &u, &v, &w);
b806      dist[u][v] = dist[v][u] = w;
968e      num[u][v] = num[v][u] = 1;
74c9  }
d52d  for(int k = 1; k <= n; k++){
076c      for(int i = 1; i <= n; i++){
979c          for(int j = 1; j <= n; j++){
5cd4              if(dist[i][k] + dist[k][j] < dist[i][j]){
096f                  dist[i][j] = dist[i][k] + dist[k][j];
10d5                  num[i][j] = num[i][k] * num[k][j];
a8fb              }
f674              else if(dist[i][k] + dist[k][j] == dist[i][j]){
ab94                  num[i][j] += num[i][k] * num[k][j];
a521              }
0d54          }
a017      }
bd7d  }
2f58  for(int i = 1; i <= n; i++){
42b2      double ans = 0;
687b      for(int s = 1; s <= n; s++){
fcd5          for(int t = 1; t <= n; t++){
30dc              if(s == i || t == i) continue;
eb87              long long numerator = 0, denominator = 0;
9793              if(dist[s][t] == dist[s][i] + dist[i][t])
bd66                  numerator += (num[s][i] * num[i][t]);
8265              denominator += (num[s][t]);
a71e              if(denominator != 0) ans += (numerator * 1.0 / denominator);
f545          }
2fd7      }
7e5b      printf("%.3f\n", ans);
db8f  }
7ea1 }
```

4.9.2 ShortestPathDAG-hashed

```
     // Ans[i] = Shortest Path from S to D when you remove edge (i)
8c39 #include <bits/stdc++.h>
3a00 using namespace std;

d426 const int MAXE = 50005;
822d const int MAXN = 7005;
c859 const int INF = (int)(1e9);

     // Information about Edges
3836 int U[MAXE], V[MAXE], W[MAXE], VAL[MAXE], E[MAXE];

     // Information about Nodes
3245 int DIST[2][MAXN], LEV[MAXN], DSU[MAXN], PAR[MAXN], F[MAXN];

     // ANS[i] = Shortest Path when you remove edge number (i)
38ae int ANS[MAXE];

2fef map < int , int > ID[MAXN]; // MAP[u][v] = Edge number of the edge (u->v)
d268 vector < int > edgeList[MAXN]; // List of Edges
9584 vector < int > tree[MAXN]; // Stores any Shortest Path Tree

9759 int N, M, Q, S, D;

f5b6 bool coolEdge[MAXE];
df41 bool coolNode[MAXN];
```

```
1781 inline void dijkstra(int src, int idx){

1477     for(int i = 0 ; i < N ; i++) DIST[idx][i] = INF;

597c     set < pair < int , pair < int , int > > > nodes;
eac3     nodes.insert( make_pair(0 , make_pair(src, 0) ) );

95e5     while(!nodes.empty()){

ddc4         int u = (*nodes.begin()).second.first;
2880         int edgeNo = (*nodes.begin()).second.second;
ed4d         int v = (U[edgeNo] + V[edgeNo]) - (u);
cc15         int c = (*nodes.begin()).first;

e441         nodes.erase(nodes.begin());

5da0         if( DIST[idx][u] == INF ){

                 // Add to Shortest Path Tree
3168             if( (idx == 0) and (edgeNo > 0) ){
e651                 coolEdge[edgeNo] = true;
cf35                 tree[u].push_back(edgeNo);
f56c                 tree[v].push_back(edgeNo);
67d6             }

f175             DIST[idx][u] = c;
55b0             for(int i = 0 ; i < edgeList[u].size() ; i++){
fb34                 edgeNo = edgeList[u][i];
ed71                 v = (U[edgeNo] + V[edgeNo]) - (u);
d8b0                 nodes.insert(make_pair(W[edgeNo] + c, make_pair(v, edgeNo)));
86b8             }
c448         }
465f     }

007d }

bdae bool dfs(int u, int p){

b385     bool inPath = (u == D);

b39f     for(int i = 0 ; i < tree[u].size() ; i++){

9e2c         int edgeNo = tree[u][i];
38db         int v = (U[edgeNo] + V[edgeNo]) - (u);
```

```
5f9b         if(edgeNo == p) continue;

f7e1         DSU[v] = PAR[v] = u;
5093         LEV[v] = LEV[u] + 1;

c919         inPath |= dfs(v, edgeNo);
ce35     }

e5dd     if(p != -1) coolEdge[p] = inPath;

4cb8     coolNode[u] = inPath;
319f     return inPath;
f9f1 }

83da bool cmp(int x , int y){
cec8     return (VAL[x]) < (VAL[y]);
460b }

813d int find(int x){
bf7c     if(coolNode[x]) return x;
6d8d     return DSU[x] = find(DSU[x]);
6b11 }

7d28 int goUP(int u, int lca, int val){

c704     if(LEV[u] <= LEV[lca]) return u;
8eca     if(!coolNode[u]) return DSU[u] = goUP(DSU[u], lca, val); // Visit each edge once

eb42     coolNode[u] = false;
         // Mark this node on Shortest Path as processed

75b6     int p = PAR[u];
ee4b     int edgeNo = ID[min(u,p)][max(u,p)];
b070     ANS[edgeNo] = val;

45d1     return DSU[u] = goUP(DSU[u], lca, val); // Compress Tree
8a53 }

0ed2 int main() {

d30e     cin.tie(0), ios::sync_with_stdio(false);

e3eb     cin >> N >> M >> Q;
64fb     for(int i = 1 ; i <= M ; i++){
```

```
d8d5        cin >> U[i] >> V[i] >> W[i];
9973        if(U[i] > V[i]) swap(U[i], V[i]);

0558        edgeList[U[i]].push_back(i);
5018        edgeList[V[i]].push_back(i);
8a0c        ID[U[i]][V[i]] = i;

ad67        ANS[i] = INF;
3d16    }

4ecb    S = 0, D = N - 1;

44cd    dijkstra(S, 0); // Dijkstra from Source
019f    dijkstra(D, 1); // Dijkstra from Destination

0572    int elen = 0;

        // Cool edges are those that are in the Shortest Path Tree
910a    for(int i = 1 ; i <= M ; i++){
ef11        if(coolEdge[i]) continue;
            // If edge is in the Shortest Path Tree, Ignore!
66ca        E[ elen++ ] = i;
ec4f        VAL[i] = min(DIST[0][U[i]] + DIST[1][V[i]], DIST[0][V[i]] + DIST[1][U[i]]) + W[i];
7f1f    }


bea3    sort(E, E + elen, cmp);
4784    dfs(S, -1);

        // Now Cool edges are those which lie on the Shortest Path from S -> D
        // Cool nodes are those which lie on the Shortest Path from S -> D

910a    for(int i = 1 ; i <= M ; i++){
5f8f        if(!coolEdge[i]) // Ans for all these edges is = Shortest Path from S -> D
93ab            ANS[i] = DIST[0][D];
5ce0    }

dde2    DSU[S] = PAR[S] = S;

642d    for(int i = 0 ; i < N ; i++){
2e23        if(!coolNode[i]){
                // If (i) isn't in Shortest Path
                // Find the first ancestor of (i) which is in Shortest Path
7e1c            DSU[i] = find(DSU[i]);
d9c4            F[i] = DSU[i];
26b3        }
0d64        else F[i] = i;
2016    }

5509    for(int i = 0 ; i < elen ; i++){

72a9        int edgeNo = E[i];
95d2        int u = U[edgeNo], v = V[edgeNo], w = VAL[edgeNo];

9b02        u = F[u], v = F[v];
2edf        int lca = (LEV[u] < LEV[v]) ? u : v;

ad06        goUP(u, lca, w);
b4b2        goUP(v, lca, w);
95d8    }

6ca1    while(Q--){
dbef        int edge;
2220        cin >> edge;
15e6        edge++;
70b4        if(ANS[edge] == INF) cout << "-1" << '\n';
daac        else cout << ANS[edge] << '\n';
66a8    }

b0c3    return 0;
1dfb }
```

5 Strings

5.1 Hashing

5.1.1 Hashing-hashed

```
     // Some Codeforces problem
8c39 #include <bits/stdc++.h>
3a00 using namespace std;
```

```
ccb1  const int MAX = 5050;
2c24  const int MOD1 = 1000000007;
c9f4  const int MOD2 = 1000000009;
e050  const int BASE = 137;

1dd6  int n, q, dp[MAX][MAX];
6aaa  char str[MAX];
27b0  pair < int, int > h[MAX], rh[MAX], p[MAX];

6424  inline int prod1(int x, int y){
1a7a    long long res = x * 1LL * y;
b496    if(res >= MOD1) res %= MOD1;
12f5    return res;
a8ea  }

6414  inline int prod2(int x, int y){
1a79    long long res = x * 1LL * y;
ca96    if(res >= MOD2) res %= MOD2;
0d75    return res;
a816  }

6c44  inline int add1(int x, int y){
d5a3    int res = x + y;
e98c    if(res < 0) res += MOD1;
6ac8    if(res >= MOD1) res -= MOD1;
a562    return res;
1556  }

6c74  inline int add2(int x, int y){
b5a3    int res = x + y;
318c    if(res < 0) res += MOD2;
6fc8    if(res >= MOD2) res -= MOD2;
a422    return res;
155c  }

      // Build tables
bcbf  void build(){
24e1    p[0] = {1, 1};
af7c    for(int i = 1; i < MAX; i++){
e9a1            p[i].first = prod1(p[i - 1].first, BASE);
6a08            p[i].second = prod2(p[i - 1].second, BASE);
432d    }
1158    h[0] = {0, 0};
3d8e    for(int i = 1; i <= n; i++){
5658            h[i].first = add1(prod1(h[i - 1].first, BASE), str[i] -
     ’a’ + 1);
05c7            h[i].second = add2(prod2(h[i - 1].second, BASE), str[i] -
     ’a’ + 1);
3853    }
3507    rh[n + 1] = {0, 0};
aeec    for(int i = n; i >= 1; i--){
47b7            rh[i].first = add1(prod1(rh[i + 1].first, BASE), str[i] -
     ’a’ + 1);
3797            rh[i].second = add2(prod2(rh[i + 1].second, BASE), str[i]
     - ’a’ + 1);
b9c1    }
0db3  }

      // Returns hash of the substring [l, r]
e9e9  pair < int, int > getHash(int l, int r){
5fb8    pair < int, int > ans = {0, 0};
fedf    ans.first = add1(h[r].first, -(prod1(h[l - 1].first, p[r - l +
     1].first)));
fac6    ans.second = add2(h[r].second,-(prod2(h[l - 1].second, p[r - l +
     1].second)));
0387    return ans;
3861  }

      // Returns hash of the substring [r, l]
ebba  pair < int, int > getReverseHash(int l, int r){
3ff2    pair < int, int > ans = {0, 0};
70f6    ans.first = add1(rh[l].first, -(prod1(rh[r + 1].first, p[r - l +
     1].first)));
a0e3    ans.second = add2(rh[l].second,-(prod2(rh[r + 1].second, p[r - l +
     1].second)));
550e    return ans;
72d5  }

ed3f  bool isPalindrome(int i, int j){
77e4    return getHash(i, j) == getReverseHash(i, j);
23c2  }

0ed2  int main(){

d481    scanf("%s", str + 1);
7ce9    n = strlen(str + 1);
8cf6    scanf("%d", &q);

f7ac    build();
d3a6   dp[n][n] = 1;
aca8   for(int i = n - 1; i >= 1; i--){
12ff          dp[i][i] = 1;
b787          dp[i][i + 1] = 2 + (str[i] == str[i + 1]);
34c7          for(int j = i + 2; j <= n; j++){
c70d                 dp[i][j] = dp[i + 1][j] + dp[i][j - 1] - dp[i +
    1][j - 1];
9f4f                 dp[i][j] += isPalindrome(i, j);
7c87          }
3b99   }

6ca3   while(q--){
a7e1          int xx, yy;
6146          scanf("%d %d", &xx, &yy);
5237          printf("%d\n", dp[xx][yy]);
baec   }
65aa }
```

5.2 KMP

5.2.1 KMP-hashed

```
8c39 #include <bits/stdc++.h>
3a00 using namespace std;


     // ------ KMP Template -------

399b const int MAX_LEN = 1e5 + 5;
d05b int lps[MAX_LEN];

     // lps[] table is 1 based, strings are 0 based.
8dab inline void compute_table(string &pattern) {
0c54   lps[0] = -1, lps[1] = 0;
0923   int pref = 0;
15c8   for (int i = 2; i <= pattern.size(); i++) {
8469          while (pref != -1 && pattern[i - 1] != pattern[pref]) {
3c21                 pref = lps[pref];
099c          }
bbc3          pref++;
2add          lps[i] = pref;
e92b   }
5f34 }

     // Function returns frequency of 'pattern' in 'text'
a0c4 inline int kmp(string &text, string &pattern){
63f8   compute_table(pattern);
ebb6   int pref = 0, count = 0;
d03f   for (int i = 0; i < text.size(); i++) {
5735          while (pref != -1 && text[i] != pattern[pref]) {
57bb                 pref = lps[pref];
dac0          }
21a8          pref++;
e94d          if (pref == pattern.size()) {
98ac                 pref = lps[pref];
45bc                 count++;
e250          }
876f   }
9f23   return count;
1c84 }

     // ---- End of KMP Template ----

0ed2 int main() {
6106   string text, pattern;
5b35   while (cin >> text >> pattern) {
6eb9          cout << kmp(text, pattern) << '\n';
cb08   }
4625 }
```

5.2.2 KMPDP-hashed

```
782b /*

f04e   Hackerearth - Benny and Two Strings
0ee9   Maximise occruences of pattern in text by modifying text following
    some constraints.
a7ee   KMP + DP problem

502e */

8c39 #include <bits/stdc++.h>
3a00 using namespace std;

62fd const int N = 205;
```

```
95f8  const int K = 505;

7ca2  string text, pattern;
e92e  int n, m, k;
49f4  int f[N][26], dp[N][N][K], lps[N];

e29c  inline int cost(int a, int b) {
e9f7      if (a > b) {
fbcd          swap(a, b);
6fa3      }
fc2d      int o1 = b - a;
9af7      int o2 = a + 26 - b;
8b12      return min(o1, o2);
9425  }

782b  /*

0e5e    dp[u][l][k] = The best I can do at position (u), current match length (l), cost remaining (k)
413f    l = x implies that in the currently built string, the last x characters are the same as the
3924    first x characters of "pattern".

502e  */

7ca5  inline int solve(int u, int l, int k) {
64ed      if (k < 0) {
ba21          return -1e9;
0dac      }
e5b5      if (u == text.size()) {
bb70          return l == (int)pattern.size();
85a6      }
dcef      if (dp[u][l][k] != -1) {
7d20          return dp[u][l][k];
0394      }
d7a0      int ans = -1e9;
9208      for (int i = 0; i < 26; ++i) {
12e2          ans = max(ans, (l == (int)pattern.size()) + solve(u + 1, f[l][i], k - cost(text[u] - ’a’, i)));
10ea      }
d4bf      return dp[u][l][k] = ans;
fed8  }

782b  /*
bb94    f[i][j] = If I have matched the first "i" characters of "pattern", and I append
59c8              character "j", what will be the new match length (lps) of the resulting string

397d    pattern => "ababa"
6019    f[3][b] = 4 --> This means I had an "aba" and I appended a "b", now the new match length is "abab"
38f7    f[3][a] = 1 --> This means I had an "aba" and I appended an "a", now the new match length is "a"

502e  */

3aad  inline void precompute() {
ddce      for (int i = 1; i < pattern.size(); ++i) {
e036          int j = lps[i - 1];
9710          while (j > 0 and pattern[j] != pattern[i]) {
a4bb              j = lps[j - 1];
dd58          }
cc9b          j += pattern[i] == pattern[j];
eb58          lps[i] = j;
c727      }
5159      for (int j = 0; j < 26; ++j) {
41b3          f[0][j] = (pattern[0] - ’a’) == j ? 1 : 0;
9a70      }
30c8      for (int i = 1; i < pattern.size(); ++i) {
a6a2          for (int j = 0; j < 26; ++j) {
2be8              f[i][j] = (pattern[i] - ’a’) == j ? i + 1 : f[lps[i - 1]][j];
4122          }
1274      }
f8b3      for (int j = 0; j < 26; ++j) {
4713          f[pattern.size()][j] = f[lps[(int)pattern.size() - 1]][j];
9a45      }
2caf  }

0ed2  int main() {
3485      cin >> n >> m >> k;
787a      cin >> pattern >> text;
6a86      precompute();
13d4      memset(dp, -1, sizeof dp);
44e1      cout << solve(0, 0, k) << ’\n’;
0a5a  }
```

5.3 Suffix Arrays

5.3.1 DistinctSubstringsSuffixArray-hashed

```
     // SPOJ Distinct Substrings - len log^2 len

8c39 #include <bits/stdc++.h>
3a00 using namespace std;

40e5 const int N = 5e4 + 5;
aea7 const int LN = 18;

0292 char str[N];

d458 /*----------- Suffix Array Template ------------*/
     // sa[i] = index of i'th smallest suffix in str[]
     // "ana" --> {a, ana, na} -> sa[0] = 2, sa[1] = 0, sa[2] = 1

2fc0 int pos[LN][N], sa[N], tmp[N];
03b1 int gap, len, level;

     // Comparison function -> O(1)
5b17 inline bool suffix_cmp(int i, int j){
66c7   if(pos[level][i] != pos[level][j]){
e07b           return (pos[level][i] < pos[level][j]);
df7e   }
6c70   i += gap, j += gap;
5dcd   if(i < len && j < len){
43cb           return (pos[level][i] < pos[level][j]);
5a63   }
e3fa   return (i > j);
d762 }

     // Builds suffix array in len log^2 len
d78c inline void build_suffix_array(){
1586   len = strlen(str);
e328   level = 0;
1197   for(int i = 0; i < len; i++){
30f1           pos[level][i] = str[i];
6afe           sa[i] = i;
f32a   }
a44b   for(gap = 1; ; gap *= 2){
a964           sort(sa, sa + len, suffix_cmp);
38b6           for(int i = 1; i < len; i++){
896e                   tmp[i] = tmp[i - 1] + suffix_cmp(sa[i - 1], sa[i]);
7436           }
2572           level = level + 1;
08fc           for(int i = 0; i < len; i++){
2367                   pos[level][sa[i]] = tmp[i];
3966           }
f1c0           if(tmp[len - 1] == len - 1) break;
07f3   }
9842 }

     // Returns LCP of str[x..len-1] and str[y..len-1] in O(log len)
0a73 inline int lcp(int x, int y){
1ddf   int res = 0;
6c54   for(int i = level; i >= 0; i--){
4331           if(x < len && y < len && pos[i][x] == pos[i][y]){
95b3                   res += (1 << i);
e78a                    x  += (1 << i);
bb0c                    y  += (1 << i);
65a5           }
2b50   }
b504   return res;
25d5 }

84f4 /*-------------- End of Template --------------*/


c3c9 inline void compute(){
6a9f   long long ans = len - sa[0];
3759   for(int i = 1; i < len; i++){
03bf           ans += len - sa[i];
a8da           ans -= lcp(sa[i - 1], sa[i]);
d53b   }
4c21   printf("%lld\n", ans);
0a1c }

0ed2 int main(){
44e5   int t;
8c31   scanf("%d", &t);
5fbb   while(t--){
a1c3           scanf("%s", str);
6070           build_suffix_array();
edbe           compute();
f710   }
87c5 }
```

5.3.2 HiddenPasswordSuffixArray-hashed

```
     // ACM ICPC - Hidden Password

8c39 #include <bits/stdc++.h>
3a00 using namespace std;

4367 const int N = 2e5 + 5;
e669 const int LN = 20;

0292 char str[N];

d458 /*----------- Suffix Array Template ------------*/
     // sa[i] = index of i'th smallest suffix in str[]
     // "ana" --> {a, ana, na} -> sa[0] = 2, sa[1] = 0, sa[2] = 1

2fc0 int pos[LN][N], sa[N], tmp[N];
03b1 int gap, len, level;

     // Comparison function -> O(1)
5b17 inline bool suffix_cmp(int i, int j){
66c7   if(pos[level][i] != pos[level][j]){
e07b          return (pos[level][i] < pos[level][j]);
df7e   }
6c70   i += gap, j += gap;
5dcd   if(i < len && j < len){
43cb          return (pos[level][i] < pos[level][j]);
5a63   }
e3fa   return (i > j);
d762 }

     // Builds suffix array in len log^2 len
d78c inline void build_suffix_array(){
1586   len = strlen(str);
e328   level = 0;
1197   for(int i = 0; i < len; i++){
30f1          pos[level][i] = str[i];
6afe          sa[i] = i;
f32a   }
a44b   for(gap = 1; ; gap *= 2){
a964          sort(sa, sa + len, suffix_cmp);
38b6          for(int i = 1; i < len; i++){
896e                 tmp[i] = tmp[i - 1] + suffix_cmp(sa[i - 1], sa[i]);
7436          }
2572          level = level + 1;
08fc          for(int i = 0; i < len; i++){
2367                 pos[level][sa[i]] = tmp[i];
3966          }
f1c0          if(tmp[len - 1] == len - 1) break;
07f3   }
9842 }

     // Returns LCP of str[x..len-1] and str[y..len-1] in O(log len)
0a73 inline int lcp(int x, int y){
1ddf   int res = 0;
6c54   for(int i = level; i >= 0; i--){
4331          if(x < len && y < len && pos[i][x] == pos[i][y]){
95b3                 res += (1 << i);
e78a                  x  += (1 << i);
bb0c                  y  += (1 << i);
65a5          }
2b50   }
b504   return res;
25d5 }

ff16 /*----------- End of Template ------------*/


e4f4 /* Returns the lexicographically smallest x length substring of
    str[]
56be    In case of multiple options, it chooses the one which has the
    lowest start_index */

baaf inline void compute(int x){
33ab   int st_idx = 0, idx = 0;
5398   for(int i = 0; i < len; i++){
6fec          if(len - sa[i] >= x){
a42b                 st_idx = sa[i];
a3a0                 idx = i;
df37                 break;
be84          }
2589   }
191a   for(int i = idx + 1; i < len; i++){
c563          if(lcp(st_idx, sa[i]) >= x)
3b57                 st_idx = min(st_idx, sa[i]);
b9a7   }
d35d   printf("%d\n", st_idx);
eee7 }

0ed2 int main(){
```

```
44e5   int t;
8c31   scanf("%d", &t);
5fbb   while(t--){
fb9b           scanf("%d %s", &len, str);
32d3           for(int i = len; i < len * 2; i++) str[i] = str[i - len];
e839           build_suffix_array();
0cc7           compute(len >> 1);
381b   }
d9bd }
```

5.4 Z Function

5.4.1 SubstringFrequencyZFunction-hashed

```
f425 #include "bits/stdc++.h"
3d1e using namespace std;

5ae1 const int N = 1e6 + 6;

5d9b char a[N], b[N], str[N * 2];
548a int t, z[N * 2];

9ee8 inline void z_function(int n){
bc68   memset(z, 0, sizeof z);
9a73   for(int i = 1, l = 0, r = 0; i < n; i++){
959c           if(i <= r) z[i] = min(z[i - l], r - i + 1);
a4bd           while(i < n && str[z[i]] == str[i + z[i]]) ++z[i];
b5f3           if(i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
9dd2   }
9493 }

0ed2 int main(){
eb0b   freopen("ioi.in", "r", stdin);
71e4   scanf("%d", &t);
2a1e   for(int qq = 1; qq <= t; qq++){
bd8b           scanf("%s %s", a, b);
2ee6           int len1 = strlen(a), len2 = strlen(b);
7c4c           for(int i = 0; i < len2; i++) str[i] = b[i];
fbcc           str[len2] = '$';
a71d           for(int i = 0; i < len1; i++) str[i + len2 + 1] = a[i];
86b5           z_function(len1 + len2 + 1);
c26b           int ans = 0;
5982           for(int i = len2 + 1; i < len2 + len1 + 1; i++) ans += (z[i] == len2);
2914           printf("Case %d: %d\n", qq, ans);
a135   }
ad74 }
```

5.4.2 TemplateZFunction-hashed

```
     // POI - Template

782b /*
8a03   Print smallest length string which can be "stamped" multiple times to get
5c25   Target String T

74c5   Suppose T = ababbababbabababbabababbababbaba
cd3d   Ans = 8, Ans_String = ababbaba

c6e2   The answer string will always be a prefix of T. Hence, you can compute z[i] for
62a9   each index (i). Now you can do an offline algorithm to solve the problem.
fa76 */

8c39 #include <bits/stdc++.h>
3a00 using namespace std;

623c const int N = 5e5 + 50;
7a56 char str[N];
3f63 int n, z[N];
30e6 vector < pair < int, int > > values;

fb9e struct node{
0e05   int mn, mx, ret;
0119   node(int _mn = -1, int _mx = -1, int _ret = -INT_MAX){
ad52           mn  = _mn;
63db           mx  = _mx;
9a1e           ret = _ret;
f4ad   }
e327 }tree[N * 4];

c9e5 inline node merge(node x, node y){
4292   if(x.mn == -1 && x.mx == -1) return y;
d4be   if(y.mn == -1 && y.mx == -1) return x;
5dfc   return node(x.mn, y.mx, max(x.ret, max(y.ret, y.mn - x.mx)));
e292 }

eba5 inline void update(int i, int l, int r, int pos){
4600   if(l == r){
56e5          tree[i].mn = tree[i].mx = l;
b5c1          return;
0dd3   }
1646   int mid = l + r >> 1;
a928   if(mid >= pos) update(i * 2, l, mid, pos);
b643   else update(i * 2 + 1, mid + 1, r, pos);
ea82   tree[i] = merge(tree[i * 2], tree[i * 2 + 1]);
1729 }

dfbb inline void z_function(){
b4f5   n = strlen(str);
13a3   for(int i = 1, l = 0, r = 0; i < n; i++){
af8d          if(i <= r) z[i] = min(z[i - l], r - i + 1);
7434          while(i < n && str[z[i]] == str[i + z[i]]) ++z[i];
5db7          if(i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
ba90   }
44fb   z[0] = n;
da5a }

0ed2 int main(){
f0aa   scanf("%s", str);
cbf8   z_function();
e812   for(int i = 0; i < n; i++){
ff4c          if(z[i]) values.push_back(make_pair(z[i], i));
6787   }
c0a5   sort(values.begin(), values.end());
baec   reverse(values.begin(), values.end());
64a9   for(int i = 0; i < n * 4; i++) tree[i] = node();
228e   int ans = n;
36c9   update(1, 0, n + 1, 0), update(1, 0, n + 1, n + 1);
edaa   for(int i = 0; i < (int) values.size(); i++){
b2e4          int cur = values[i].first, j = i + 1;
1a91          while((j < (int) values.size()) && (values[j].first ==
    cur)) j++;
7207          j--;
023e          for(int k = i; k <= j; k++){
300b                 int idx = 1 + values[k].second;
9154                 update(1, 0, n + 1, idx);
a4f7          }
fa06          if(tree[1].ret <= cur) ans = cur;
300a          i = j;
51fd   }
396e   printf("%d\n", ans);
71b6 }
```