

Contents

1 Misc	1	4.3.1 Formal Power Series	14
1.1 Contest	1	4.4 Theorems	15
1.1.1 Makefile	1	4.4.1 Kirchhoff's Theorem	15
1.2 How Did We Get Here?	1	4.4.2 Tutte's Matrix	15
1.2.1 Fast I/O	1	4.4.3 Cayley's Formula	15
1.3 Tools	2	4.4.4 Erdős–Gallai Theorem	16
1.3.1 Floating Point Binary Search	2	4.4.5 Burnside's Lemma	16
1.3.2 SplitMix64	2		
1.3.3 <random>	2		
1.3.4 x86 Stack Hack	2		
1.4 Algorithms	2	5 Numeric	16
1.4.1 Bit Hacks	2	5.1 Barrett Reduction	16
1.4.2 Aliens Trick	2	5.2 Long Long Multiplication	16
1.4.3 Hilbert Curve	2	5.3 Fast Fourier Transform	16
1.4.4 Infinite Grid Knight Distance	2	5.4 Fast Walsh-Hadamard Transform	16
1.4.5 Poker Hand	3	5.5 Subset Convolution	16
1.4.6 Longest Increasing Subsequence	3	5.6 Linear Recurrences	16
1.4.7 Mo's Algorithm on Tree	3	5.6.1 Berlekamp-Massey Algorithm	16
2 Data Structures	3	5.6.2 Linear Recurrence Calculation	16
2.1 Segment Tree (ZKW)	3	5.7 Matrices	17
2.2 Line Container	3	5.7.1 Determinant	17
2.3 Li-Chao Tree	3	5.7.2 Inverse	17
2.4 Heavy-Light Decomposition	4	5.7.3 Characteristic Polynomial	17
2.5 Wavelet Matrix	4	5.7.4 Solve Linear Equation	18
2.6 Link-Cut Tree	5	5.8 Polynomial Interpolation	18
3 Graph	6	5.9 Simplex Algorithm	18
3.1 Modeling	6	6 Geometry	19
3.2 Matching/Flows	6	6.1 Point	19
3.2.1 Dinic's Algorithm	6	6.1.1 Quaternion	19
3.2.2 Minimum Cost Flow	6	6.1.2 Spherical Coordinates	19
3.2.3 Gomory-Hu Tree	7	6.2 Segments	19
3.2.4 Global Minimum Cut	7	6.3 Convex Hull	19
3.2.5 Bipartite Minimum Cover	7	6.3.1 3D Hull	20
3.2.6 Edmonds' Algorithm	7	6.4 Angular Sort	20
3.2.7 Minimum Weight Matching	8	6.5 Convex Polygon Minkowski Sum	20
3.2.8 Stable Marriage	8	6.6 Point In Polygon	20
3.2.9 Kuhn-Munkres algorithm	9	6.6.1 Convex Version	20
3.3 Shortest Path Faster Algorithm	9	6.7 Closest Pair	21
3.4 Strongly Connected Components	9	6.8 Minimum Enclosing Circle	21
3.4.1 2-Satisfiability	10	6.9 Delaunay Triangulation	21
3.5 Biconnected Components	10	6.9.1 Slower Version	21
3.5.1 Articulation Points	10	6.10 Half Plane Intersection	22
3.5.2 Bridges	10		
3.6 Triconnected Components	10		
3.7 Centroid Decomposition	10		
3.8 Minimum Mean Cycle	11		
3.9 Directed MST	11		
3.10 Maximum Clique	11		
3.11 Dominator Tree	12	7 Strings	22
3.12 Manhattan Distance MST	12	7.1 Knuth-Morris-Pratt Algorithm	22
4 Math	12	7.2 Aho-Corasick Automaton	22
4.1 Number Theory	12	7.3 Suffix Array	22
4.1.1 Mod Struct	12	7.4 Suffix Tree	23
4.1.2 Miller-Rabin	12	7.5 Cocke-Younger-Kasami Algorithm	23
4.1.3 Linear Sieve	12	7.6 Z Value	23
4.1.4 Get Factors	12	7.7 Manacher's Algorithm	24
4.1.5 Binary GCD	12	7.8 Minimum Rotation	24
4.1.6 Extended GCD	12	7.9 Palindromic Tree	24
4.1.7 Chinese Remainder Theorem	12		
4.1.8 Baby-Step Giant-Step	12		
4.1.9 Pollard's Rho	13		
4.1.10 Tonelli-Shanks Algorithm	13		
4.1.11 Chinese Sieve	13		
4.1.12 Rational Number Binary Search	13		
4.1.13 Farey Sequence	14		
4.2 Combinatorics	14	8 Debug List	24
4.2.1 Matroid Intersection	14		
4.2.2 De Bruijn Sequence	14		
4.2.3 Multinomial	14		
4.3 Algebra	14		

```

9 public class fast_io {
11    public static PrintWriter out =
12        new PrintWriter(new BufferedOutputStream(System.out));
13    static FASTIO in = new FASTIO();
15
16    public static void main(String[] args) throws IOException {
17        int cp = in.nextInt();
18        while (cp-- > 0) {
19            solve();
20        }
21        out.close();
22    }
23
24    static void solve() {
25
26        static class FASTIO {
27            BufferedReader br;
28            StringTokenizer st;
29
30            public FASTIO() {
31                br = new BufferedReader(
32                    new InputStreamReader(System.in)
33                );
34
35                String next() {
36                    while (st == null || !st.hasMoreElements()) {
37                        try {
38                            st = new StringTokenizer(br.readLine());
39                        } catch (IOException e) {
40                            e.printStackTrace();
41                        }
42                    }
43                    return st.nextToken();
44                }
45
46                int nextInt() {
47                    return Integer.parseInt(next());
48                }
49
50                long nextLong() {
51                    return Long.parseLong(next());
52                }
53
54                double nextDouble() {
55                    return Double.parseDouble(next());
56                }
57
58                String nextLine() {
59                    String str = "";
60                    try {
61                        st = null;
62                        str = br.readLine();
63                    } catch (IOException e) {
64                        e.printStackTrace();
65                    }
66                    return str;
67                }
68
69            }
70        }
71    }

```

1.3. Tools

1.3.1. Floating Point Binary Search

```

1 union di {
2     double d;
3     ull i;
4 };
5 bool check(double);
// binary search in [L, R) with relative error 2^-eps
6 double binary_search(double L, double R, int eps) {
7     di l = {L}, r = {R}, m;
8     while (r.i - l.i > 1LL << (52 - eps)) {
9         m.i = (l.i + r.i) >> 1;
10        if (check(m.d)) r = m;
11        else l = m;
12    }
13    return l.d;
14 }

```

1.3.2. SplitMix64

```

1 using ull = unsigned long long;
2 inline ull splitmix64(ull x) {
3     // change to static ull x = SEED; ` for DRBG
4     ull z = (x += 0x9E3779B97F4A7C15);
5     z = (z ^ (z >> 30)) * 0xBF58476D1CE4E5B9;
6     z = (z ^ (z >> 27)) * 0x94D049BB13311EB;
7     return z ^ (z >> 31);
}

```

1.3.3. <random>

```

1 import java.util.Random;
2
3 class random {
4     static final Random rng = new Random();
5
6     static int randInt(int l, int r) {
7         return l + rng.nextInt(r - l + 1);
8     }
9
10    static long randLong(long l, long r) {
11        return l + (Math.abs(rng.nextLong()) % (r - l + 1));
12    }
13    // use inside the main
14    // int a = randInt(1, 10);
15    // long b = randLong(100, 1000);
16 }

```

1.3.4. x86 Stack Hack

```

1 constexpr size_t size = 200 << 20; // 200MiB
2 int main() {
3     register long rsp asm("rsp");
4     char *buf = new char[size];
5     asm("movq %%0, %%rsp\n" ::"r"(buf + size));
6     // do stuff
7     asm("movq %%0, %%rsp\n" ::"r"(rsp));
8     delete[] buf;
9 }

```

1.4. Algorithms

1.4.1. Bit Hacks

```

1 // next permutation of x as a bit sequence
2 ull next_bits_permutation(ull x) {
3     ull c = __builtin_ctzll(x), r = x + (1ULL << c);
4     return (r ^ x) >> (c + 2) | r;
5 }
6 // iterate over all (proper) subsets of bitset s
7 void subsets(ull s) {
8     for (ull x = s; x;) { -x &= s; /* do stuff */ }
9 }

```

1.4.2. Aliens Trick

```

1 // min dp[i] value and its i (smallest one)
2 pll get_dp(int cost);
3 ll aliens(int k, int l, int r) {
4     while (l != r) {
5         int m = (l + r) / 2;
6         auto [f, s] = get_dp(m);
7         if (s == k) return f - m * k;
8         if (s < k) r = m;
9         else l = m + 1;
10    }
11    return get_dp(l).first - l * k;
12 }

```

1.4.3. Hilbert Curve

```

1 ll hilbert(ll n, int x, int y) {
2     ll res = 0;
3     for (ll s = n; s /= 2;) {
4         int rx = !(x & s), ry = !(y & s);
5         res += s * s * ((3 * rx) ^ ry);
6         if (ry == 0) {
7             if (rx == 1) x = s - 1 - x, y = s - 1 - y;
8             swap(x, y);
9         }
10    }
11    return res;
12 }

```

1.4.4. Infinite Grid Knight Distance

```

1 ll get_dist(ll dx, ll dy) {
2     if (++(dx = abs(dx)) > ++(dy = abs(dy))) swap(dx, dy);
3     if (dx == 1 && dy == 2) return 3;
4     if (dx == 3 && dy == 3) return 4;
5     ll lb = max(dy / 2, (dx + dy) / 3);
6     return ((dx ^ dy ^ lb) & 1) ? ++lb : lb;
7 }

```

1.4.5. Poker Hand

```

1
3
5
7 using namespace std;
9
11 struct hand {
12     static constexpr auto rk = [] {
13         array<int, 256> x{};
14         auto s = "23456789TJQKACDHS";
15         for (int i = 0; i < 17; i++) x[s[i]] = i % 13;
16         return x;
17     }();
18     vector<pair<int, int>> v;
19     vector<int> cnt, vf, vs;
20     int type;
21     hand() : cnt(4), type(0) {}
22     void add_card(char suit, char rank) {
23         ++cnt[rk[suit]];
24         for (auto &f, s : v)
25             if (s == rk[rank]) return ++f, void();
26         v.emplace_back(1, rk[rank]);
27     }
28     void process() {
29         sort(v.rbegin(), v.rend());
30         for (auto [f, s] : v) vf.push_back(f), vs.push_back(s);
31         bool str = 0, flu = find(all(cnt), 5) != cnt.end();
32         if ((str = v.size()) == 5)
33             for (int i = 1; i < 5; i++)
34                 if (vs[i] != vs[i - 1] + 1) str = 0;
35         if (vs == vector<int>{12, 3, 2, 1, 0})
36             str = 1, vs = {3, 2, 1, 0, -1};
37         if (str && flu) type = 9;
38         else if (vf[0] == 4) type = 8;
39         else if (vf[0] == 3 && vf[1] == 2) type = 7;
40         else if (str || flu) type = 5 + flu;
41         else if (vf[0] == 3) type = 4;
42         else if (vf[0] == 2) type = 2 + (vf[1] == 2);
43         else type = 1;
44     }
45     bool operator<(const hand &b) const {
46         return make_tuple(type, vf, vs) <
47             make_tuple(b.type, b.vf, b.vs);
48     }

```

1.4.6. Longest Increasing Subsequence

```

1
3 template <class I> vi lis(const vector<I> &S) {
4     if (S.empty()) return {};
5     vi prev(sz(S));
6     typedef pair<I, int> p;
7     vector<p> res;
8     rep(i, 0, sz(S)) {
9         // change 0 -> i for longest non-decreasing subsequence
10        auto it = lower_bound(all(res), p{S[i], 0});
11        if (it == res.end())
12            res.emplace_back(), it = res.end() - 1;
13        *it = {S[i], i};
14        prev[i] = it == res.begin() ? 0 : (it - 1)->second;
15    }
16    int L = sz(res), cur = res.back().second;
17    vi ans(L);
18    while (L--) ans[L] = cur, cur = prev[cur];
19    return ans;
}

```

1.4.7. Mo's Algorithm on Tree

```

1 void MoAlgoOnTree() {
2     Dfs(0, -1);
3     vector<int> euler(tk);
4     for (int i = 0; i < n; ++i) {
5         euler[tin[i]] = i;
6         euler[tout[i]] = i;
7     }
8     vector<int> l(q), r(q), qr(q), sp(q, -1);
9     for (int i = 0; i < q; ++i) {
10        if (tin[u[i]] > tin[v[i]]) swap(u[i], v[i]);
11        int z = GetLCA(u[i], v[i]);
12        sp[i] = z[i];
13        if (z == u[i]) l[i] = tin[u[i]], r[i] = tin[v[i]];
14        else l[i] = tout[u[i]], r[i] = tin[v[i]];
15        qr[i] = i;
16    }
17    sort(qr.begin(), qr.end(), [&](int i, int j) {
18        if (l[i] / kB == l[j] / kB) return r[i] < r[j];
19        return l[i] / kB < l[j] / kB;
}

```

```

21     });
22     vector<bool> used(n);
23     // Add(v): add/remove v to/from the path based on used[v]
24     for (int i = 0, tl = 0, tr = -1; i < q; ++i) {
25         while (tl < l[qr[i]]) Add(euler[tl++]);
26         while (tl > l[qr[i]]) Add(euler[--tl]);
27         while (tr > r[qr[i]]) Add(euler[tr--]);
28         while (tr < r[qr[i]]) Add(euler[++tr]);
29     }
}

```

2. Data Structures

2.1. Segment Tree (ZKW)

```

1 struct segtree {
2     using T = int;
3     T f(T a, T b) { return a + b; } // any monoid operation
4     static constexpr T ID = 0; // identity element
5     int n;
6     vector<T> v;
7     segtree(int n_) : n(n_), v(2 * n, ID) {}
8     segtree(vector<T> &a) : n(a.size()), v(2 * n, ID) {
9         copy_n(a.begin(), n, v.begin() + n);
10        for (int i = n - 1; i > 0; i--)
11            v[i] = f(v[i * 2], v[i * 2 + 1]);
12    }
13    void update(int i, T x) {
14        for (v[i += n] = x; i /= 2; )
15            v[i] = f(v[i * 2], v[i * 2 + 1]);
16    }
17    T query(int l, int r) {
18        T tl = ID, tr = ID;
19        for (l += n, r += n; l < r; l /= 2, r /= 2) {
20            if (l & 1) tl = f(tl, v[l++]);
21            if (r & 1) tr = f(v[--r], tr);
22        }
23        return f(tl, tr);
24    }
25 }

```

2.2. Line Container

```

1
3 struct Line {
4     mutable ll k, m, p;
5     bool operator<(const Line &o) const { return k < o.k; }
6     bool operator<(ll x) const { return p < x; }
7 };
8 // add: line y=kx+m, query: maximum y of given x
9 struct LineContainer : multiset<Line, less<> {
10     // (for doubles, use inf = 1./0, div(a,b) = a/b)
11     static const ll inf = LLONG_MAX;
12     ll div(ll a, ll b) { // floored division
13         return a / b - ((a ^ b) < 0 && a % b); }
14     bool isect(iterator x, iterator y) {
15         if (y == end()) return x->p = inf, 0;
16         if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
17         else x->p = div(y->m - x->m, x->k - y->k);
18         return x->p >= y->p;
19     }
20     void add(ll k, ll m) {
21         auto z = insert({k, m, 0}), y = z++, x = y;
22         while (isect(y, z)) z = erase(z);
23         if (x != begin() && isect(--x, y))
24             isect(x, y = erase(y));
25         while ((y = x) != begin() && (--x)->p >= y->p)
26             isect(x, erase(y));
27     }
28     ll query(ll x) {
29         assert(!empty());
30         auto l = *lower_bound(x);
31         return l.k * x + l.m;
32     }
}

```

2.3. Li-Chao Tree

```

1 public class LiChaoTree {
2
3     // Represents a line y = mx + c
4     static class Line {
5         long m, c;
6
7         public Line(long m, long c) {
8             this.m = m;
9             this.c = c;
10        }
11    }
}

```

```

// Evaluates the line at a given x-coordinate
long eval(long x) {
    return m * x + c;
}

// Node of the Li-Chao Tree
static class Node {
    Line line;
    Node left, right;

    public Node(Line line) {
        this.line = line;
    }
}

private Node root;
private final long minCoord;
private final long maxCoord;
private final Line identityLine; // Represents "no line" or infinity for queries

// Constructor for the Li-Chao Tree
// minCoord and maxCoord define the range of x-values the tree will handle.
// identityLine should return a very large value for min queries (or very small for max queries)
public LiChaoTree(long minCoord, long maxCoord) {
    this.minCoord = minCoord;
    this.maxCoord = maxCoord;
    // For minimum queries, an identity line should return a very large value.
    // Using Long.MAX_VALUE for 'c' and 0 for 'm' ensures it's always "worse" than any real line.
    this.identityLine = new Line(0, Long.MAX_VALUE);
    this.root = new Node(identityLine);
}

// Adds a new line to the tree
public void addLine(Line.newLine) {
    addLine(root, minCoord, maxCoord, newLine);
}

private void addLine(Node node, long currentMin, long currentMax) {
    long mid = currentMin + (currentMax - currentMin) / 2;
    boolean leftBetter = newLine.eval(currentMin) < node.line.eval(mid);
    boolean midBetter = newLine.eval(mid) < node.line.eval(mid);

    if (midBetter) {
        // If the new line is better at the midpoint, swap it with the current line
        Line temp = node.line;
        node.line = newLine;
        newLine = temp; // The old line now becomes the 'new'
    }

    // If the interval is a single point, we are done
    if (currentMin == currentMax) {
        return;
    }

    // Decide which child to push the 'worse' line to
    if (leftBetter != midBetter) { // Intersection point is in the left child's range
        if (node.left == null) {
            node.left = new Node(identityLine);
        }
        addLine(node.left, currentMin, mid, newLine);
    } else if (leftBetter == midBetter && leftBetter == false) { // Intersection point is in the right child's range
        if (node.right == null) {
            node.right = new Node(identityLine);
        }
        addLine(node.right, mid + 1, currentMax, newLine);
    }
    // If leftBetter == midBetter == true, it means the new line is better across the whole interval
    // and the old line is completely dominated, so no need to push it down.
}

// Queries the minimum value at a given x-coordinate
public long query(long x) {
    return query(root, minCoord, maxCoord, x);
}

private long query(Node node, long currentMin, long currentMax) {
    if (node == null) {
        return identityLine.eval(x); // No line in this path
    }

    long res = node.line.eval(x);

    if (currentMin == currentMax) {
        return res; // Reached a leaf node
    }

    long mid = currentMin + (currentMax - currentMin) / 2;
    if (x <= mid) {
        res = Math.min(res, query(node.left, currentMin, mid));
    } else {
        res = Math.min(res, query(node.right, mid + 1, currentMax));
    }
    return res;
}

107
109
111
113
115
117
119
121
123
125
127
129
131
133
135
137
139
141
143
145
147
149
151
153
155
157
159
161
163
165
167
169
171
173
175
177
179
181
183
185
187
189
191
193
195
197
199
201
203
205
207
209
211
213
215
217
219
221
223
225
227
229
231
233
235
237
239
241
243
245
247
249
251
253
255
257
259
261
263
265
267
269
271
273
275
277
279
281
283
285
287
289
291
293
295
297
299
301
303
305
307
309
311
313
315
317
319
321
323
325
327
329
331
333
335
337
339
341
343
345
347
349
351
353
355
357
359
361
363
365
367
369
371
373
375
377
379
381
383
385
387
389
391
393
395
397
399
401
403
405
407
409
411
413
415
417
419
421
423
425
427
429
431
433
435
437
439
441
443
445
447
449
451
453
455
457
459
461
463
465
467
469
471
473
475
477
479
481
483
485
487
489
491
493
495
497
499
501
503
505
507
509
511
513
515
517
519
521
523
525
527
529
531
533
535
537
539
541
543
545
547
549
551
553
555
557
559
561
563
565
567
569
571
573
575
577
579
581
583
585
587
589
591
593
595
597
599
601
603
605
607
609
611
613
615
617
619
621
623
625
627
629
631
633
635
637
639
641
643
645
647
649
651
653
655
657
659
661
663
665
667
669
671
673
675
677
679
681
683
685
687
689
691
693
695
697
699
701
703
705
707
709
711
713
715
717
719
721
723
725
727
729
731
733
735
737
739
741
743
745
747
749
751
753
755
757
759
761
763
765
767
769
771
773
775
777
779
781
783
785
787
789
791
793
795
797
799
801
803
805
807
809
811
813
815
817
819
821
823
825
827
829
831
833
835
837
839
841
843
845
847
849
851
853
855
857
859
861
863
865
867
869
871
873
875
877
879
881
883
885
887
889
891
893
895
897
899
901
903
905
907
909
911
913
915
917
919
921
923
925
927
929
931
933
935
937
939
941
943
945
947
949
951
953
955
957
959
961
963
965
967
969
971
973
975
977
979
981
983
985
987
989
991
993
995
997
999
1001
1003
1005
1007
1009
1011
1013
1015
1017
1019
1021
1023
1025
1027
1029
1031
1033
1035
1037
1039
1041
1043
1045
1047
1049
1051
1053
1055
1057
1059
1061
1063
1065
1067
1069
1071
1073
1075
1077
1079
1081
1083
1085
1087
1089
1091
1093
1095
1097
1099
1101
1103
1105
1107
1109
1111
1113
1115
1117
1119
1121
1123
1125
1127
1129
1131
1133
1135
1137
1139
1141
1143
1145
1147
1149
1151
1153
1155
1157
1159
1161
1163
1165
1167
1169
1171
1173
1175
1177
1179
1181
1183
1185
1187
1189
1191
1193
1195
1197
1199
1201
1203
1205
1207
1209
1211
1213
1215
1217
1219
1221
1223
1225
1227
1229
1231
1233
1235
1237
1239
1241
1243
1245
1247
1249
1251
1253
1255
1257
1259
1261
1263
1265
1267
1269
1271
1273
1275
1277
1279
1281
1283
1285
1287
1289
1291
1293
1295
1297
1299
1301
1303
1305
1307
1309
1311
1313
1315
1317
1319
1321
1323
1325
1327
1329
1331
1333
1335
1337
1339
1341
1343
1345
1347
1349
1351
1353
1355
1357
1359
1361
1363
1365
1367
1369
1371
1373
1375
1377
1379
1381
1383
1385
1387
1389
1391
1393
1395
1397
1399
1401
1403
1405
1407
1409
1411
1413
1415
1417
1419
1421
1423
1425
1427
1429
1431
1433
1435
1437
1439
1441
1443
1445
1447
1449
1451
1453
1455
1457
1459
1461
1463
1465
1467
1469
1471
1473
1475
1477
1479
1481
1483
1485
1487
1489
1491
1493
1495
1497
1499
1501
1503
1505
1507
1509
1511
1513
1515
1517
1519
1521
1523
1525
1527
1529
1531
1533
1535
1537
1539
1541
1543
1545
1547
1549
1551
1553
1555
1557
1559
1561
1563
1565
1567
1569
1571
1573
1575
1577
1579
1581
1583
1585
1587
1589
1591
1593
1595
1597
1599
1601
1603
1605
1607
1609
1611
1613
1615
1617
1619
1621
1623
1625
1627
1629
1631
1633
1635
1637
1639
1641
1643
1645
1647
1649
1651
1653
1655
1657
1659
1661
1663
1665
1667
1669
1671
1673
1675
1677
1679
1681
1683
1685
1687
1689
1691
1693
1695
1697
1699
1701
1703
1705
1707
1709
1711
1713
1715
1717
1719
1721
1723
1725
1727
1729
1731
1733
1735
1737
1739
1741
1743
1745
1747
1749
1751
1753
1755
1757
1759
1761
1763
1765
1767
1769
1771
1773
1775
1777
1779
1781
1783
1785
1787
1789
1791
1793
1795
1797
1799
1801
1803
1805
1807
1809
1811
1813
1815
1817
1819
1821
1823
1825
1827
1829
1831
1833
1835
1837
1839
1841
1843
1845
1847
1849
1851
1853
1855
1857
1859
1861
1863
1865
1867
1869
1871
1873
1875
1877
1879
1881
1883
1885
1887
1889
1891
1893
1895
1897
1899
1901
1903
1905
1907
1909
1911
1913
1915
1917
1919
1921
1923
1925
1927
1929
1931
1933
1935
1937
1939
1941
1943
1945
1947
1949
1951
1953
1955
1957
1959
1961
1963
1965
1967
1969
1971
1973
1975
1977
1979
1981
1983
1985
1987
1989
1991
1993
1995
1997
1999
2001
2003
2005
2007
2009
2011
2013
2015
2017
2019
2021
2023
2025
2027
2029
2031
2033
2035
2037
2039
2041
2043
2045
2047
2049
2051
2053
2055
2057
2059
2061
2063
2065
2067
2069
2071
2073
2075
2077
2079
2081
2083
2085
2087
2089
2091
2093
2095
2097
2099
2101
2103
2105
2107
2109
2111
2113
2115
2117
2119
2121
2123
2125
2127
2129
2131
2133
2135
2137
2139
2141
2143
2145
2147
2149
2151
2153
2155
2157
2159
2161
2163
2165
2167
2169
2171
2173
2175
2177
2179
2181
2183
2185
2187
2189
2191
2193
2195
2197
2199
2201
2203
2205
2207
2209
2211
2213
2215
2217
2219
2221
2223
2225
2227
2229
2231
2233
2235
2237
2239
2241
2243
2245
2247
2249
2251
2253
2255
2257
2259
2261
2263
2265
2267
2269
2271
2273
2275
2277
2279
2281
2283
2285
2287
2289
2291
2293
2295
2297
2299
2301
2303
2305
2307
2309
2311
2313
2315
2317
2319
2321
2323
2325
2327
2329
2331
2333
2335
2337
2339
2341
2343
2345
2347
2349
2351
2353
2355
2357
2359
2361
2363
2365
2367
2369
2371
2373
2375
2377
2379
2381
2383
2385
2387
2389
2391
2393
2395
2397
2399
2401
2403
2405
2407
2409
2411
2413
2415
2417
2419
2421
2423
2425
2427
2429
2431
2433
2435
2437
2439
2441
2443
2445
2447
2449
2451
2453
2455
2457
2459
2461
2463
2465
2467
2469
2471
2473
2475
2477
2479
2481
2483
2485
2487
2489
2491
2493
2495
2497
2499
2501
2503
2505
2507
2509
2511
2513
2515
2517
2519
2521
2523
2525
2527
2529
2531
2533
2535
2537
2539
2541
2543
2545
2547
2549
2551
2553
2555
2557
2559
2561
2563
2565
2567
2569
2571
2573
2575
2577
2579
2581
2583
2585
2587
2589
2591
2593
2595
2597
2599
2601
2603
2605
2607
2609
2611
2613
2615
2617
2619
2621
2623
2625
2627
2629
2631
2633
2635
2637
2639
2641
2643
2645
2647
2649
2651
2653
2655
2657
2659
2661
2663
2665
2667
2669
2671
2673
2675
2677
2679
2681
2683
2685
2687
2689
2691
2693
2695
2697
2699
2701
2703
2705
2707
2709
2711
2713
2715
2717
2719
2721
2723
2725
2727
2729
2731
2733
2735
2737
2739
2741
2743
2745
2747
2749
2751
2753
2755
2757
2759
2761
2763
2765
2767
2769
2771
2773
2775
2777
2779
2781
2783
2785
2787
2789
2791
2793
2795
2797
2799
2801
2803
2805
2807
2809
2811
2813
2815
2817
2819
2821
2823
2825
2827
2829
2831
2833
2835
2837
2839
2841
2843
2845
2847
2849
2851
2853
2855
2857
2859
2861
2863
2865
2867
2869
2871
2873
2875
2877
2879
2881
2883
2885
2887
2889
2891
2893
2895
2897
2899
2901
2903
2905
2907
2909
2911
2913
2915
2917
2919
2921
2923
2925
2927
2929
2931
2933
2935
2937
2939
2941
2943
2945
2947
2949
2951
2953
2955
2957
2959
2961
2963
2965
2967
2969
2971
2973
2975
2977
2979
2981
2983
2985
2987
2989
2991
2993
2995
2997
2999
3001
3003
3005
3007
3009
3011
3013
3015
3017
3019
3021
3023
3025
3027
3029
3031
3033
3035
3037
3039
3041
3043
3045
3047
3049
3051
3053
3055
3057
3059
3061
3063
3065
3067
3069
3071
3073
3075
3077
3079
3081
3083
3085
3087
3089
3091
3093
3095
3097
3099
3101
3103
3105
3107
3109
3111
3113
3115
3117
3119
3121
3123
3125
3127
3129
3131
3133
3135
3137
3139
3141
3143
3145
3147
3149
3151
3153
3155
3157
3159
3161
3163
3165
3167
3169
3171
3173
3175
3177
3179
3181
3183
3185
3187
3189
3191
3193
3195
3197
3199
3201
3203
3205
3207
3209
3211
3213
3215
3217
3219
3221
3223
3225
3227
3229
3231
3233
3235
3237
3239
3241
3243
3245
3247
3249
3251
3253
3255
3257
3259
3261
3263
3265
3267
3269
3271
3273
3275
3277
3279
3281
3283
3285
3287
3289
3291
3293
3295
3297
3299
3301
3303
3305
3307
3309
3311
3313
3315
3317
3319
3321
3323
3325
3327
3329
3331
3333
3335
3337
3339
3341
3343
3345
3347
3349
3351
3353
3355
3357
3359
3361
3363
3365
3367
3369
3371
3373
3375
3377
3379
3381
3383
3385
3387
3389
3391
3393
3395
3397
3399
3401
3403
3405
3407
3409
3411
3413
3415
3417
3419
3421
3423
3425
3427
3429
3431
3433
3435
3437
3439
3441
3443
3445
3447
3449
3451
3453
3455
3457
3459
3461
3463
3465
3467
3469
3471
3473
3475
3477
3479
3481
3483
3485
3487
3489
3491
3493
3495
3497
3499
3501
3503
3505
3507
3509
3511
3513
3515
3517
3519
3521
3523
3525
3527
3529
3531
3533
3535
3537
3539
3541
3543
3545
3547
3549
3551
3553
3555
3557
3559
3561
3563
3565
3567
3569
3571
3573
3575
3577
3579
3581
3583
3585
3587
3589
3591
3593
3595
3597
3599
3601
3603
3605
3607
3609
3611
3613
3615
3617
3619
3621
3623
3625
3627
3629
3631
3633
3635
3637
3639
3641
3643
3645
3647
3649
3651
3653
3655
3657
3659
3661
3663
3665
3667
3669
3671
3673
3675
3677
3679
3681
3683
3685
3687
3689
3691
3693
3695
3697
3699
3701
3703
3705
3707
3709
3711
3713
3715
3717
3719
3721
3723
3725
3727
3729
3731
3733
3735
3737
3739
3741
3743
3745
3747
3749
3751
3753
3755
3757
3759
3761
3763
3765
3767
3769
3771
3773
3775
3777
3779
3781
3783
3785
3787
3789
3791
3793
3795
3797
3799
3801
3803
3805
3807
3809
3811
3813
3815
3817
3819
3821
3823
3825
3827
3829
3831
3833
3835
3837
3839
3841
3843
3845
3847
3849
3851
3853
3855
3857
3859
3861
3863
3865
3867
3869
3871
3873
3875
3877
3879
3881
3883
3885
3887
3889
3891
3893
3895
3897
3899
3901
3903
3905
3907
3909
3911
3913
3915
3917
3919
3921
3923
3925
3927
3929
3931
3933
3935
3937
3939
3941
3943
3945
3947
3949
3951
3953
3955
3957
3959
3961
3963
3965
3967
3969
3971
3973
3975
3977
3979
3981
3983
3985
3987
3989
3991
3993
3995
3997
3999
4001
4003
4005
4007
4009
4011
4013
4015
4017
4019
4021
4023
4025
4027
4029
4031
4033
4035
4037
4039
4041
4043
4045
4047
4049
4051
4053
4055
4057
4059
4061
4063
4065
4067
4069
4071
4073
4075
4077
4079
4081
4083
4085
4087
4089
4091
4093
4095
4097
4099
4101
4103
4105
4107
4109
4111
4113
4115
4117
4119
4121
4123
4125
4127
4129
4131
4133
4135
4137
4139
4141
4143
4145
4147
4149
4151
4153
4155
4157
4159
4161
4163
4165
4167
4169
4171
4173
4175
4177
4179
4181
4183
4185
4187
4189
4191
4193
4195
4197
4199
4201
4203
4205
4207
4209
4211
4213
4215
4217
4219
4221
4223
4225
4227
4229
4231
4233
4235
4237
4239
4241
4243
4245
4247
4249
4251
4253
4255
4257
4259
4261
4263
4265
4267
4269
4271
4273
4275
4277
4279
4281
4283
4285
4287
4289
4291
4293
4295
4297
4299
4301
4303
4305
4307
4309
4311
4313
4315
4317
4319
4321
4323
4325
4327
4329
4331
4333
4335
4337
4339
4341
4343
4345
4347
4349
4351
4353
4355
4357
4359
4361
4363
4365
4367
4369
4371
4373
4375
4377
4379
4381
4383
4385
4387
4389
4391
4393
4395
4397
4399
4401
4403
4405
4407
4409
4411
4413
4415
4417
4419
4421
4423
4425
4427
4429
4431
4433
4435
4437
4439
4441
4443
4445
4447
4449
4451
4453
4455
4457
4459
4461
4463
4465
4467
4469
4471
4473
4475
4477
4479
4481
4483
4485
4487
4489
4491
4493
4495
4497
4499
4501
4503
4505
4507
4509
4511
4513
4515
4517
4519
4521
4523
4525
4527
4529
4531
4533
4535
4537
4539
4541
4543
4545
4547
4549
4551
4553
4555
4557
4559
4561
4563
4565
4567
4569
4571
4573
4575
4577
4579
4581
4583
4585
4587
4589
4591
4593
4595
4597
4599
4601
4603
4605
4607
4609
4611
4613
4615
4617
4619
4621
4623
4625
4627
4629
4631
4633
4635
4637
4639
4641
4643
4645
4647
4649
```

2.4. Heavy-Light Decomposition

```

very large value.
is always "worse" than any real line.
3 template <bool VALS_EDGES> struct HLD {
4     int N, tim = 0;
5     vector<vi> adj;
6     vi par, siz, depth, rt, pos;
7     Node *tree;
8     HLD(vector<vi> adj_) :
9         : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
10        depth(N), rt(N), pos(N), tree(new Node(0, N)) {
11        dfsSz(0);
12        dfsHld(0);
13    }
14    Line newLine() {
15        void dfssz(int v) {
16            if (currLine[v] != -1)
17                adj[v].erase(find(all(adj[v]), par[v]));
18            for (int &u : adj[v]) {
19                par[u] = v, depth[u] = depth[v] + 1;
20                dfssz(u);
21                siz[v] += siz[u];
22                if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
23            }
24        }
25        void dfsHld(int v) {
26            pos[v] = tim++;
27            for (int u : adj[v]) {
28                rt[u] = (u == adj[v][0] ? rt[v] : u);
29                dfsHld(u);
30            }
31        }
32    }
33    template <class B> void process(int u, int v, B op) {
34        for (; rt[u] != rt[v]; v = par[rt[v]]) {
35            if (depth[rt[u]] > depth[rt[v]]) swap(u, v);
36            op(pos[rt[v]], pos[v] + 1);
37        }
38        if (depth[u] > depth[v]) swap(u, v);
39        op(pos[u] + VALS_EDGES, pos[v] + 1);
40    }
41    void modifyPath(int u, int v, int val) {
42        process(u, v,
43            [&](int l, int r) { tree->add(l, r, val); });
44    }
45    int queryPath(int u,
46                  int v) { // Modify depending on problem
47        int res = -1e9;
48        process(u, v, [&](int l, int r) {
49            res = max(res, tree->query(l, r));
50        });
51        return res;
52    }
53    long x) {
54        int querySubtree(int v) { // modifySubtree is similar
55            return tree->query(pos[v] + VALS_EDGES,
56                                 pos[v] + siz[v]);
57        }
58    }

```

2.5. Wavelet Matrix

```
1
2
3 #pragma GCC target("popcnt,bmi2")
4 #include <immintrin.h>
5
6 tMax,T x)) unsigned. You might want to compress values first
7 template <typename T> struct wavelet_matrix {
8     static_assert(is_unsigned_v<T>, "only unsigned T");
9 }
```

```

11 struct bit_vector {
12     static constexpr uint W = 64;
13     uint n, cnt0;
14     vector<ull> bits;
15     vector<uint> sum;
16     bit_vector(uint n_) : n(n_), bits(n / W + 1), sum(n / W + 1) {}
17     void build() {
18         for (uint j = 0; j != n / W; ++j)
19             sum[j + 1] = sum[j] + _mm_popcnt_u64(bits[j]);
20         cnt0 = rank0(n);
21     }
22     void set_bit(uint i) { bits[i / W] |= 1ULL << i % W; }
23     bool operator[](uint i) const {
24         return !(bits[i / W] & 1ULL << i % W);
25     }
26     uint rank1(uint i) const {
27         return sum[i / W] +
28             _mm_popcnt_u64(_bzhi_u64(bits[i / W], i % W));
29     }
30     uint rank0(uint i) const { return i - rank1(i); }
31 };
32 uint n, lg;
33 vector<bit_vector> b;
34 wavelet_matrix(const vector<T> &a) : n(a.size()) {
35     lg =
36         __lg(max(*max_element(a.begin(), a.end()), T(1))) + 1;
37     b.assign(lg, n);
38     vector<T> cur = a, nxt(n);
39     for (int h = lg; h--;) {
40         for (uint i = 0; i < n; ++i)
41             if (cur[i] & (T(1) << h)) b[h].set_bit(i);
42         b[h].build();
43         int il = 0, ir = b[h].cnt0;
44         for (uint i = 0; i < n; ++i)
45             nxt[(b[h][i] ? ir : il)++] = cur[i];
46         swap(cur, nxt);
47     }
48     T operator[](uint i) const {
49         T res = 0;
50         for (int h = lg; h--;) {
51             if (b[h][i])
52                 i += b[h].cnt0 - b[h].rank0(i), res |= T(1) << h;
53             else i = b[h].rank0(i);
54         }
55         return res;
56     }
57     // query k-th smallest (0-based) in a[l, r)
58     T kth(uint l, uint r, uint k) const {
59         T res = 0;
60         for (int h = lg; h--;) {
61             uint tl = b[h].rank0(l), tr = b[h].rank0(r);
62             if (k >= tr - tl) {
63                 k -= tr - tl;
64                 l += b[h].cnt0 - tl;
65                 r += b[h].cnt0 - tr;
66                 res |= T(1) << h;
67             } else l = tl, r = tr;
68         }
69         return res;
70     }
71     // count of i in [l, r) with a[i] < u
72     uint count(uint l, uint r, T u) const {
73         if (u >= T(1) << lg) return r - l;
74         uint res = 0;
75         for (int h = lg; h--;) {
76             uint tl = b[h].rank0(l), tr = b[h].rank0(r);
77             if (u & (T(1) << h)) {
78                 l += b[h].cnt0 - tl;
79                 r += b[h].cnt0 - tr;
80                 res += tr - tl;
81             } else l = tl, r = tr;
82         }
83         return res;
84     }
85 }

```

2.6. Link-Cut Tree

```

1
2
3 const int MXN = 100005;
4 const int MEM = 100005;
5
6 struct Splay {
7     static Splay nil, mem[MEM], *pmem;
8     Splay *ch[2], *f;
9     int val, rev, size;
10    Splay() : val(-1), rev(0), size(0) {
11        f = ch[0] = ch[1] = &nil;
12    }
13    Splay(int _val) : val(_val), rev(0), size(1) {
14        f = ch[0] = ch[1] = &nil;
15    }

```

```

17     bool isr() {
18         return f->ch[0] != this && f->ch[1] != this;
19     }
20     int dir() { return f->ch[0] == this ? 0 : 1; }
21     void setCh(Splay *c, int d) {
22         ch[d] = c;
23         if (c != &nil) c->f = this;
24         pull();
25     }
26     void push() {
27         if (rev) {
28             swap(ch[0], ch[1]);
29             if (ch[0] != &nil) ch[0]->rev ^= 1;
30             if (ch[1] != &nil) ch[1]->rev ^= 1;
31             rev = 0;
32         }
33     }
34     void pull() {
35         size = ch[0]->size + ch[1]->size + 1;
36         if (ch[0] != &nil) ch[0]->f = this;
37         if (ch[1] != &nil) ch[1]->f = this;
38     }
39 } Splay::nil, Splay::mem[MEM], *Splay::pmem = Splay::mem;
40 Splay *nil = &Splay::nil;
41
42 void rotate(Splay *x) {
43     Splay *p = x->f;
44     int d = x->dir();
45     if (!p->isr()) p->f->setCh(x, p->dir());
46     else x->f = p->f;
47     p->setCh(x->ch[d], d);
48     x->setCh(p, !d);
49     p->pull();
50     x->pull();
51 }
52
53 vector<Splay *> splayVec;
54 void splay(Splay *x) {
55     splayVec.clear();
56     for (Splay *q = x;; q = q->f) {
57         splayVec.push_back(q);
58         if (q->isr()) break;
59     }
60     reverse(begin(splayVec), end(splayVec));
61     for (auto it : splayVec) it->push();
62     while (!x->isr()) {
63         if (x->f->isr()) rotate(x);
64         else if (x->dir() == x->f->dir())
65             rotate(x->f), rotate(x);
66         else rotate(x), rotate(x);
67     }
68 }
69
70 Splay *access(Splay *x) {
71     Splay *q = nil;
72     for (; x != nil; x = x->f) {
73         splay(x);
74         x->setCh(q, 1);
75         q = x;
76     }
77     return q;
78 }
79 void evert(Splay *x) {
80     access(x);
81     splay(x);
82     x->rev ^= 1;
83     x->push();
84     x->pull();
85 }
86 void link(Splay *x, Splay *y) {
87     // evert(x);
88     access(x);
89     splay(x);
90     evert(y);
91     x->setCh(y, 1);
92 }
93 void cut(Splay *x, Splay *y) {
94     // evert(x);
95     access(y);
96     splay(y);
97     y->push();
98     y->ch[0] = y->ch[0]->f = nil;
99 }
100
101 int N, Q;
102 Splay *vt[MXN];
103
104 int ask(Splay *x, Splay *y) {
105     access(x);
106     access(y);
107     splay(x);
108     int res = x->f->val;
109     if (res == -1) res = x->val;
110     return res;
111 }

```

```

111 }
112 int main(int argc, char **argv) {
113     scanf("%d", &N, &Q);
114     for (int i = 1; i <= N; i++)
115         vt[i] = new (Splay::pmem++) Splay(i);
116     while (Q--) {
117         char cmd[105];
118         int u, v;
119         scanf("%s", cmd);
120         if (cmd[1] == 'i') {
121             scanf("%d%d", &u, &v);
122             link(vt[u], vt[v]);
123         } else if (cmd[0] == 'c') {
124             scanf("%d", &v);
125             cut(vt[1], vt[v]);
126         } else {
127             scanf("%d%d", &u, &v);
128             int res = ask(vt[u], vt[v]);
129             printf("%d\n", res);
130         }
131     }

```

3. Graph

3.1. Modeling

- Maximum/Minimum flow with lower bound / Circulation problem
 1. Construct super source S and sink T .
 2. For each edge (x, y, l, u) , connect $x \rightarrow y$ with capacity $u - l$.
 3. For each vertex v , denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
 4. If $in(v) > 0$, connect $S \rightarrow v$ with capacity $in(v)$, otherwise, connect $v \rightarrow T$ with capacity $-in(v)$.
 - To maximize, connect $t \rightarrow s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T . If $f \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
 - To minimize, let f be the maximum flow from S to T . Connect $t \rightarrow s$ with capacity ∞ and let the flow from S to T be f' . If $f + f' \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, f' is the answer.
 5. The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.
- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)
 1. Redirect every edge: $y \rightarrow x$ if $(x, y) \in M$, $x \rightarrow y$ otherwise.
 2. DFS from unmatched vertices in X .
 3. $x \in X$ is chosen iff x is unvisited.
 4. $y \in Y$ is chosen iff y is visited.
- Minimum cost cyclic flow
 1. Construct super source S and sink T
 2. For each edge (x, y, c) , connect $x \rightarrow y$ with $(cost, cap) = (c, 1)$ if $c > 0$, otherwise connect $y \rightarrow x$ with $(cost, cap) = (-c, 1)$
 3. For each edge with $c < 0$, sum these cost as K , then increase $d(y)$ by 1, decrease $d(x)$ by 1
 4. For each vertex v with $d(v) > 0$, connect $S \rightarrow v$ with $(cost, cap) = (0, d(v))$
 5. For each vertex v with $d(v) < 0$, connect $v \rightarrow T$ with $(cost, cap) = (0, -d(v))$
 6. Flow from S to T , the answer is the cost of the flow $C + K$
- Maximum density induced subgraph
 1. Binary search on answer, suppose we're checking answer T
 2. Construct a max flow model, let K be the sum of all weights
 3. Connect source $s \rightarrow v$, $v \in G$ with capacity K
 4. For each edge (u, v, w) in G , connect $u \rightarrow v$ and $v \rightarrow u$ with capacity w
 5. For $v \in G$, connect it with sink $v \rightarrow t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
 6. T is a valid answer if the maximum flow $f < K|V|$
- Minimum weight edge cover
 1. For each $v \in V$ create a copy v' , and connect $u' \rightarrow v'$ with weight $w(u, v)$.
 2. Connect $v \rightarrow v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v .
 3. Find the minimum weight perfect matching on G' .
- Project selection problem
 1. If $p_v > 0$, create edge (s, v) with capacity p_v ; otherwise, create edge (v, t) with capacity $-p_v$.
 2. Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v .
 3. The mincut is equivalent to the maximum profit of a subset of projects.
- 0/1 quadratic programming

$$\sum_x c_{xx} + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x \bar{y} + \sum_{xyx'y'} c_{xyx'y'} (x \bar{y} + x' \bar{y}')$$

can be minimized by the mincut of the following graph:

1. Create edge (x, t) with capacity c_x and create edge (s, y) with capacity c_y .
2. Create edge (x, y) with capacity c_{xy} .
3. Create edge (x, y) and edge (x', y') with capacity $c_{xyx'y'}$.

3.2. Matching/Flows

3.2.1. Dinic's Algorithm

```

1 struct Dinic {
2     struct edge {
3         int to, cap, flow, rev;
4     };
5     static constexpr int MAXN = 1000, MAXF = 1e9;
6     vector<edge> v[MAXN];
7     int top[MAXN], deep[MAXN], side[MAXN], s, t;
8     void make_edge(int s, int t, int cap) {
9         v[s].push_back({t, cap, 0, (int)v[t].size()});
10        v[t].push_back({s, 0, 0, (int)v[s].size() - 1});
11    }
12    int dfs(int a, int flow) {
13        if (a == t || !flow) return flow;
14        for (int &i = top[a]; i < v[a].size(); i++) {
15            edge &e = v[a][i];
16            if (deep[a] + 1 == deep[e.to] && e.cap - e.flow) {
17                int x = dfs(e.to, min(e.cap - e.flow, flow));
18                if (x) {
19                    e.flow += x, v[e.to][e.rev].flow -= x;
20                    return x;
21                }
22            }
23        }
24        deep[a] = -1;
25        return 0;
26    }
27    bool bfs() {
28        queue<int> q;
29        fill_n(deep, MAXN, 0);
30        q.push(s), deep[s] = 1;
31        int tmp;
32        while (!q.empty()) {
33            tmp = q.front(), q.pop();
34            for (edge e : v[tmp])
35                if (!deep[e.to] && e.cap != e.flow)
36                    deep[e.to] = deep[tmp] + 1, q.push(e.to);
37        }
38        return deep[t];
39    }
40    int max_flow(int _s, int _t) {
41        s = _s, t = _t;
42        int flow = 0, tflow;
43        while (bfs())
44            fill_n(deep, MAXN, 0);
45        while ((tflow = dfs(s, MAXF))) flow += tflow;
46    }
47    return flow;
48}
49 void reset() {
50    fill_n(side, MAXN, 0);
51    for (auto &i : v) i.clear();
52}
53

```

3.2.2. Minimum Cost Flow

```

1 struct MCF {
2     struct edge {
3         ll to, from, cap, flow, cost, rev;
4     } *fromE[MAXN];
5     vector<edge> v[MAXN];
6     ll n, s, t, flows[MAXN], dis[MAXN], pi[MAXN], flowlim;
7     void make_edge(int s, int t, ll cap, ll cost) {
8         if (!cap) return;
9         v[s].pb(edge{t, s, cap, 0LL, cost, v[t].size()});
10        v[t].pb(edge{s, t, 0LL, 0LL, -cost, v[s].size() - 1});
11    }
12    bitset<MAXN> vis;
13    void dijkstra() {
14        vis.reset();
15        __gnu_pbds::priority_queue<pair<ll, int>> q;
16        vector<decltype(q)::point_iterator> its(n);
17        q.push({0LL, s});
18        while (!q.empty()) {
19            int now = q.top().second;
20            q.pop();
21            if (vis[now]) continue;
22            vis[now] = 1;
23            ll ndis = dis[now] + pi[now];
24            for (edge &e : v[now]) {
25                if (e.flow == e.cap || vis[e.to]) continue;
26                if (dis[e.to] > ndis + e.cost - pi[e.to]) {
27                    dis[e.to] = ndis + e.cost - pi[e.to];
28                    flows[e.to] = min(flows[now], e.cap - e.flow);
29                    fromE[e.to] = &e;
30                }
31            }
32        }
33    }
34

```

```

31     if (its[e.to] == q.end())
32         its[e.to] = q.push({-dis[e.to], e.to});
33     else q.modify(its[e.to], {-dis[e.to], e.to});
34 }
35 }
36
37 bool AP(ll &flow) {
38     fill_n(dis, n, INF);
39     fromE[s] = 0;
40     dis[s] = 0;
41     flows[s] = flowlim - flow;
42     dijkstra();
43     if (dis[t] == INF) return false;
44     flow += flows[t];
45     for (edge *e = fromE[t]; e; e = fromE[e->from]) {
46         e->flow += flows[t];
47         v[e->to][e->rev].flow -= flows[t];
48     }
49     for (int i = 0; i < n; i++)
50         pi[i] = min(pi[i] + dis[i], INF);
51     return true;
52 }
53 pll solve(int _s, int _t, ll _flowlim = INF) {
54     s = _s, t = _t, flowlim = _flowlim;
55     pll re;
56     while (re.F != flowlim && AP(re.F));
57     for (int i = 0; i < n; i++)
58         for (edge &e : v[i])
59             if (e.flow != 0) re.S += e.flow * e.cost;
60     re.S /= 2;
61     return re;
62 }
63 void init(int _n) {
64     n = _n;
65     fill_n(pi, n, 0);
66     for (int i = 0; i < n; i++) v[i].clear();
67 }
68 void setpi(int s) {
69     fill_n(pi, n, INF);
70     pi[s] = 0;
71     for (ll it = 0, flag = 1, tdis; flag && it < n; it++) {
72         flag = 0;
73         for (int i = 0; i < n; i++)
74             if (pi[i] != INF)
75                 for (edge &e : v[i])
76                     if (e.cap && (tdis = pi[i] + e.cost) < pi[e.to])
77                         pi[e.to] = tdis, flag = 1;
78     }
79 }

```

3.2.3. Gomory-Hu Tree

Requires: Dinic's Algorithm

```

1
2 int e[MAXN][MAXN];
3 int p[MAXN];
4 Dinic D; // original graph
5 void gomory_hu() {
6     fill(p, p + n, 0);
7     fill(e[0], e[n], INF);
8     for (int s = 1; s < n; s++) {
9         int t = p[s];
10        Dinic F = D;
11        int tmp = F.max_flow(s, t);
12        for (int i = 1; i < s; i++)
13            e[s][i] = e[i][s] = min(tmp, e[t][i]);
14        for (int i = s + 1; i <= n; i++)
15            if (p[i] == t && F.side[i]) p[i] = s;
16    }
17 }

```

3.2.4. Global Minimum Cut

```

1
2 // weights is an adjacency matrix, undirected
3 pair<int, vi> getMinCut(vector<vi> &weights) {
4     int N = sz(weights);
5     vi used(N), cut, best_cut;
6     int best_weight = -1;
7
8     for (int phase = N - 1; phase >= 0; phase--) {
9         vi w = weights[0], added = used;
10        int prev, k = 0;
11        rep(i, 0, phase) {
12            prev = k;
13            k = -1;
14            rep(j, 1, N) if (!added[j] &&
15                           (k == -1 || w[j] > w[k])) k = j;
16            if (i == phase - 1) {
17

```

```

19             rep(j, 0, N) weights[prev][j] += weights[k][j];
20             rep(j, 0, N) weights[j][prev] = weights[prev][j];
21             used[k] = true;
22             cut.push_back(k);
23             if (best_weight == -1 || w[k] < best_weight) {
24                 best_cut = cut;
25                 best_weight = w[k];
26             } else {
27                 rep(j, 0, N) w[j] += weights[k][j];
28                 added[k] = true;
29             }
30         }
31     }
32     return {best_weight, best_cut};
33 }

```

3.2.5. Bipartite Minimum Cover

Requires: Dinic's Algorithm

```

1
2 // maximum independent set = all vertices not covered
3 // x : [0, n), y : [0, m]
4 struct Bipartite_vertex_cover {
5     Dinic D;
6     int n, m, s, t, x[maxn], y[maxn];
7     void make_edge(int x, int y) { D.make_edge(x, y + n, 1); }
8     int matching() {
9         int re = D.max_flow(s, t);
10        for (int i = 0; i < n; i++)
11            for (Dinic::edge &e : D.v[i])
12                if (e.to != s && e.flow == 1) {
13                    x[i] = e.to - n, y[e.to - n] = i;
14                    break;
15                }
16        return re;
17    }
18    // init() and matching() before use
19    void solve(vector<int> &vx, vector<int> &vy) {
20        bitset<maxn * 2 + 10> vis;
21        queue<int> q;
22        for (int i = 0; i < n; i++)
23            if (x[i] == -1) q.push(i), vis[i] = 1;
24        while (!q.empty()) {
25            int now = q.front();
26            q.pop();
27            if (now < n) {
28                for (Dinic::edge &e : D.v[now])
29                    if (e.to != s && e.to - n != x[now] && !vis[e.to])
30                        vis[e.to] = 1, q.push(e.to);
31            } else {
32                if (!vis[y[now - n]])
33                    vis[y[now - n]] = 1, q.push(y[now - n]);
34            }
35        }
36        for (int i = 0; i < n; i++)
37            if (!vis[i]) vx.pb(i);
38        for (int i = 0; i < m; i++)
39            if (vis[i + n]) vy.pb(i);
40    }
41    void init(int _n, int _m) {
42        n = _n, m = _m, s = n + m, t = s + 1;
43        for (int i = 0; i < n; i++)
44            x[i] = -1, D.make_edge(s, i, 1);
45        for (int i = 0; i < m; i++)
46            y[i] = -1, D.make_edge(i + n, t, 1);
47    }
48 }

```

3.2.6. Edmonds' Algorithm

```

1
2 struct Edmonds {
3     int n, T;
4     vector<vector<int>> g;
5     vector<int> pa, p, used, base;
6     Edmonds(int n)
7         : n(n), T(0), g(n), pa(n, -1), p(n), used(n),
8          base(n) {}
9     void add(int a, int b) {
10        g[a].push_back(b);
11        g[b].push_back(a);
12    }
13    int getBase(int i) {
14        while (i != base[i])
15            base[i] = base[base[i]], i = base[i];
16        return i;
17    }
18    vector<int> toJoin;
19    void mark_path(int v, int x, int b, vector<int> &path) {
20        for (; getBase(v) != b; v = p[x]) {
21

```

```

23     p[v] = x, x = pa[v];
24     toJoin.push_back(v);
25     toJoin.push_back(x);
26     if (!used[x]) used[x] = ++T, path.push_back(x);
27 }
28 bool go(int v) {
29     for (int x : g[v]) {
30         int b, bv = getBase(v), bx = getBase(x);
31         if (bv == bx) {
32             continue;
33         } else if (used[x]) {
34             vector<int> path;
35             toJoin.clear();
36             if (used[bx] < used[bv])
37                 mark_path(v, x, b = bx, path);
38             else mark_path(x, v, b = bv, path);
39             for (int z : toJoin) base[getBase(z)] = b;
40             for (int z : path)
41                 if (go(z)) return 1;
42         } else if (p[x] == -1) {
43             p[x] = v;
44             if (pa[x] == -1) {
45                 for (int y; x != -1; x = v)
46                     y = p[x], v = pa[y], pa[x] = y, pa[y] = x;
47                 return 1;
48             }
49             if (!used[pa[x]]) {
50                 used[pa[x]] = ++T;
51                 if (go(pa[x])) return 1;
52             }
53         }
54     }
55     return 0;
56 }
57 void init_dfs() {
58     for (int i = 0; i < n; i++)
59         used[i] = 0, p[i] = -1, base[i] = i;
60 }
61 bool dfs(int root) {
62     used[root] = ++T;
63     return go(root);
64 }
65 void match() {
66     int ans = 0;
67     for (int v = 0; v < n; v++)
68         for (int x : g[v])
69             if (pa[v] == -1 && pa[x] == -1) {
70                 pa[v] = x, pa[x] = v, ans++;
71                 break;
72             }
73     init_dfs();
74     for (int i = 0; i < n; i++)
75         if (pa[i] == -1 && dfs(i)) ans++, init_dfs();
76     cout << ans * 2 << "\n";
77     for (int i = 0; i < n; i++)
78         if (pa[i] > i)
79             cout << i + 1 << " " << pa[i] + 1 << "\n";
80 }
81 }

```

3.2.7. Minimum Weight Matching

```

1 struct Graph {
2     static const int MAXN = 105;
3     int n, e[MAXN][MAXN];
4     int match[MAXN], d[MAXN], onstk[MAXN];
5     vector<int> stk;
6     void init(int _n) {
7         n = _n;
8         for (int i = 0; i < n; i++)
9             for (int j = 0; j < n; j++)
10                // change to appropriate infinity
11                // if not complete graph
12                e[i][j] = 0;
13     }
14     void add_edge(int u, int v, int w) {
15         e[u][v] = e[v][u] = w;
16     }
17     bool SPFA(int u) {
18         if (onstk[u]) return true;
19         stk.push_back(u);
20         onstk[u] = 1;
21         for (int v = 0; v < n; v++) {
22             if (u != v && match[u] != v && !onstk[v]) {
23                 int m = match[v];
24                 if (d[m] > d[u] - e[v][m] + e[u][v]) {
25                     d[m] = d[u] - e[v][m] + e[u][v];
26                     onstk[v] = 1;
27                     stk.push_back(v);
28                     if (SPFA(m)) return true;
29                     stk.pop_back();
30                     onstk[v] = 0;
31                 }
32             }
33         }
34     }
35 }
36 
```

```

33     }
34     onstk[u] = 0;
35     stk.pop_back();
36     return false;
37 }
38 int solve() {
39     for (int i = 0; i < n; i += 2) {
40         match[i] = i + 1;
41         match[i + 1] = i;
42     }
43     while (true) {
44         int found = 0;
45         for (int i = 0; i < n; i++) onstk[i] = d[i] = 0;
46         for (int i = 0; i < n; i++) {
47             stk.clear();
48             if (!onstk[i] && SPFA(i)) {
49                 found = 1;
50                 while (stk.size() >= 2) {
51                     int u = stk.back();
52                     stk.pop_back();
53                     int v = stk.back();
54                     stk.pop_back();
55                     match[u] = v;
56                     match[v] = u;
57                 }
58             }
59         }
60         if (!found) break;
61     }
62     int ret = 0;
63     for (int i = 0; i < n; i++) ret += e[i][match[i]];
64     ret /= 2;
65     return ret;
66 }
67 } graph;

```

3.2.8. Stable Marriage

```

1 // normal stable marriage problem
2 /* input:
3  *
4  * Albert Laura Nancy Marcy
5  * Brad Marcy Nancy Laura
6  * Chuck Laura Marcy Nancy
7  * Laura Chuck Albert Brad
8  * Marcy Albert Chuck Brad
9  * Nancy Brad Albert Chuck
10 */
11
12 using namespace std;
13 const int MAXN = 505;
14
15 int n;
16 int favor[MAXN][MAXN]; // favor[boy_id][rank] = girl_id;
17 int order[MAXN][MAXN]; // order[girl_id][boy_id] = rank;
18 int current[MAXN]; // current[boy_id] = rank;
19 // boy_id will pursue current[boy_id] girl.
20 int girl_current[MAXN]; // girl[girl_id] = boy_id;
21
22 void initialize() {
23     for (int i = 0; i < n; i++) {
24         current[i] = 0;
25         girl_current[i] = n;
26         order[i][n] = n;
27     }
28 }
29
30 map<string, int> male, female;
31 string bname[MAXN], gname[MAXN];
32 int fit = 0;
33
34 void stable_marriage() {
35     queue<int> que;
36     for (int i = 0; i < n; i++) que.push(i);
37     while (!que.empty()) {
38         int boy_id = que.front();
39         que.pop();
40         int girl_id = favor[boy_id][current[boy_id]];
41         current[boy_id]++;
42         if (order[girl_id][boy_id] <
43             order[girl_id][girl_current[girl_id]]) {
44             if (girl_current[girl_id] < n)
45                 que.push(girl_current[girl_id]);
46             girl_current[girl_id] = boy_id;
47         } else {
48             que.push(boy_id);
49         }
50     }
51 }
52
53 }
```

```

57 int main() {
58     cin >> n;
59
60     for (int i = 0; i < n; i++) {
61         string p, t;
62         cin >> p;
63         male[p] = i;
64         bname[i] = p;
65         for (int j = 0; j < n; j++) {
66             cin >> t;
67             if (!female.count(t)) {
68                 gname[fit] = t;
69                 female[t] = fit++;
70             }
71             favor[i][j] = female[t];
72         }
73     }
74
75     for (int i = 0; i < n; i++) {
76         string p, t;
77         cin >> p;
78         for (int j = 0; j < n; j++) {
79             cin >> t;
80             order[female[p]][male[t]] = j;
81         }
82     }
83
84     initialize();
85     stable_marriage();
86
87     for (int i = 0; i < n; i++) {
88         cout << bname[i] << "\n";
89         << gname[favor[i][current[i] - 1]] << endl;
90     }
91 }

```

3.2.9. Kuhn-Munkres algorithm

```

1 // Maximum Weight Perfect Bipartite Matching
2 // Detect non-perfect-matching:
3 // 1. set all edge[i][j] as INF
4 // 2. if solve() >= INF, it is not perfect matching.
5
6 typedef long long ll;
7 struct KM {
8     static const int MAXN = 1050;
9     static const ll INF = 1LL << 60;
10    int n, match[MAXN], vx[MAXN], vy[MAXN];
11    ll edge[MAXN][MAXN], lx[MAXN], ly[MAXN], slack[MAXN];
12    void init(int _n) {
13        n = _n;
14        for (int i = 0; i < n; i++)
15            for (int j = 0; j < n; j++) edge[i][j] = 0;
16    }
17    void add_edge(int x, int y, ll w) { edge[x][y] = w; }
18    bool DFS(int x) {
19        vx[x] = 1;
20        for (int y = 0; y < n; y++) {
21            if (vy[y]) continue;
22            if ((lx[x] + ly[y] > edge[x][y])) {
23                slack[y] =
24                    min(slack[y], lx[x] + ly[y] - edge[x][y]);
25            } else {
26                vy[y] = 1;
27                if (match[y] == -1 || DFS(match[y])) {
28                    match[y] = x;
29                    return true;
30                }
31            }
32        }
33        return false;
34    }
35    ll solve() {
36        fill(match, match + n, -1);
37        fill(lx, lx + n, -INF);
38        fill(ly, ly + n, 0);
39        for (int i = 0; i < n; i++)
40            for (int j = 0; j < n; j++)
41                lx[i] = max(lx[i], edge[i][j]);
42        for (int i = 0; i < n; i++) {
43            fill(slack, slack + n, INF);
44            while (true) {
45                fill(vx, vx + n, 0);
46                fill(vy, vy + n, 0);
47                if (DFS(i)) break;
48                ll d = INF;
49                for (int j = 0; j < n; j++)
50                    if (!vy[j]) d = min(d, slack[j]);
51                for (int j = 0; j < n; j++) {
52                    if (vx[j]) lx[j] -= d;
53                    if (vy[j]) ly[j] += d;
54                    else slack[j] -= d;
55                }
56            }
57        }
58    }
59
60    int main() {
61        cin >> n;
62
63        for (int i = 0; i < n; i++) {
64            string p, t;
65            cin >> p;
66            male[p] = i;
67            bname[i] = p;
68            for (int j = 0; j < n; j++) {
69                cin >> t;
70                if (!female.count(t)) {
71                    gname[fit] = t;
72                    female[t] = fit++;
73                }
74                favor[i][j] = female[t];
75            }
76        }
77
78        for (int i = 0; i < n; i++) {
79            string p, t;
80            cin >> p;
81            for (int j = 0; j < n; j++) {
82                cin >> t;
83                order[female[p]][male[t]] = j;
84            }
85        }
86
87        initialize();
88        stable_marriage();
89
90        for (int i = 0; i < n; i++) {
91            cout << bname[i] << "\n";
92            << gname[favor[i][current[i] - 1]] << endl;
93        }
94    }
95 }

```

```

57     }
58 }
59     ll res = 0;
60     for (int i = 0; i < n; i++) {
61         res += edge[match[i]][i];
62     }
63 }
64 graph;

```

3.3. Shortest Path Faster Algorithm

```

1 struct SPFA {
2     static const int maxn = 1010, INF = 1e9;
3     int dis[maxn];
4     bitset<maxn> inq, inneg;
5     queue<int> q, tq;
6     vector<pii> v[maxn];
7     void make_edge(int s, int t, int w) {
8         v[s].emplace_back(t, w);
9     }
10    void dfs(int a) {
11        inneg[a] = 1;
12        for (pii i : v[a])
13            if (!inneg[i.F]) dfs(i.F);
14    }
15    bool solve(int n, int s) { // true if have neg-cycle
16        for (int i = 0; i <= n; i++) dis[i] = INF;
17        dis[s] = 0, q.push(s);
18        for (int i = 0; i < n; i++) {
19            inq.reset();
20            int now;
21            while (!q.empty()) {
22                now = q.front(), q.pop();
23                for (pii &i : v[now]) {
24                    if (dis[i.F] > dis[now] + i.S) {
25                        dis[i.F] = dis[now] + i.S;
26                        if (!inq[i.F]) tq.push(i.F), inq[i.F] = 1;
27                    }
28                }
29            }
30            q.swap(tq);
31        }
32        bool re = !q.empty();
33        inneg.reset();
34        while (!q.empty()) {
35            if (!inneg[q.front()]) dfs(q.front());
36            q.pop();
37        }
38        return re;
39    }
40    void reset(int n) {
41        for (int i = 0; i <= n; i++) v[i].clear();
42    }
43 }

```

3.4. Strongly Connected Components

```

1 struct TarjanScc {
2     int n, step;
3     vector<int> time, low, instk, stk;
4     vector<vector<int>> e, scc;
5     TarjanScc(int n_) :
6         n(n_), step(0), time(n), low(n), instk(n), e(n) {}
7     void add_edge(int u, int v) { e[u].push_back(v); }
8     void dfs(int x) {
9         time[x] = low[x] = ++step;
10        stk.push_back(x);
11        instk[x] = 1;
12        for (int y : e[x])
13            if (!time[y]) {
14                dfs(y);
15                low[x] = min(low[x], low[y]);
16            } else if (instk[y]) {
17                low[x] = min(low[x], time[y]);
18            }
19        if (time[x] == low[x]) {
20            scc.emplace_back();
21            for (int y = -1; y != x;) {
22                y = stk.back();
23                stk.pop_back();
24                instk[y] = 0;
25                scc.back().push_back(y);
26            }
27        }
28    }
29    void solve() {
30        for (int i = 0; i < n; i++)
31            if (!time[i]) dfs(i);
32        reverse(scc.begin(), scc.end());
33        // scc in topological order
34    }
35 }

```

3.4.1. 2-Satisfiability

Requires: Strongly Connected Components

```

1
3 // 1 based, vertex in SCC = MAXN * 2
// (not i) is i + n
5 struct two_SAT {
    int n, ans[MAXN];
    SCC S;
    void imply(int a, int b) { S.make_edge(a, b); }
    bool solve(int _n) {
        n = _n;
        S.solve(n * 2);
        for (int i = 1; i <= n; i++) {
            if (S.scc[i] == S.scc[i + n]) return false;
            ans[i] = (S.scc[i] < S.scc[i + n]);
        }
        return true;
    }
    void init(int _n) {
        n = _n;
        fill_n(ans, n + 1, 0);
        S.init(n * 2);
    }
} SAT;

```

3.5. Biconnected Components

3.5.1. Articulation Points

```

1 void dfs(int x, int p) {
    tin[x] = low[x] = ++t;
    int ch = 0;
    for (auto u : g[x])
        if (u.first != p) {
            if (!ins[u.second])
                st.push(u.second), ins[u.second] = true;
            if (tin[u.first]) {
                low[x] = min(low[x], tin[u.first]);
                continue;
            }
            ++ch;
            dfs(u.first, x);
            low[x] = min(low[x], low[u.first]);
            if (low[u.first] >= tin[x]) {
                cut[x] = true;
                ++sz;
                while (true) {
                    int e = st.top();
                    st.pop();
                    bcc[e] = sz;
                    if (e == u.second) break;
                }
            }
        }
    if (ch == 1 && p == -1) cut[x] = false;
}

```

3.5.2. Bridges

```

1 // if there are multi-edges, then they are not bridges
void dfs(int x, int p) {
    tin[x] = low[x] = ++t;
    st.push(x);
    for (auto u : g[x])
        if (u.first != p) {
            if (tin[u.first]) {
                low[x] = min(low[x], tin[u.first]);
                continue;
            }
            dfs(u.first, x);
            low[x] = min(low[x], low[u.first]);
            if (low[u.first] == tin[u.first]) br[u.second] = true;
        }
    if (tin[x] == low[x]) {
        ++sz;
        while (st.size())
            int u = st.top();
            st.pop();
            bcc[u] = sz;
            if (u == x) break;
    }
}

```

3.6. Triconnected Components

```

1
3 // requires a union-find data structure
struct ThreeEdgeCC {

```

```

7     int v, ind;
    vector<int> id, pre, post, low, deg, path;
    vector<vector<int>> components;
    UnionFind uf;
    template <class Graph>
    void dfs(const Graph &G, int v, int prev) {
        pre[v] = ++ind;
        for (int w : G[v])
            if (w != v) {
                if (w == prev) {
                    prev = -1;
                    continue;
                }
                if (pre[w] != -1) {
                    if (pre[w] < pre[v]) {
                        deg[v]++;
                        low[v] = min(low[v], pre[w]);
                    } else {
                        deg[v]--;
                        int &u = path[v];
                        for (; u != -1 && pre[u] <= pre[w] &&
                            pre[w] <= post[u]); {
                            uf.join(v, u);
                            deg[v] += deg[u];
                            u = path[u];
                        }
                    }
                    continue;
                }
            }
        dfs(G, w, v);
        if (path[w] == -1 && deg[w] <= 1) {
            deg[v] += deg[w];
            low[v] = min(low[v], low[w]);
            continue;
        }
        if (deg[w] == 0) w = path[w];
        if (low[v] > low[w]) {
            low[v] = min(low[v], low[w]);
            swap(w, path[v]);
        }
        for (; w != -1; w = path[w]) {
            uf.join(v, w);
            deg[v] += deg[w];
        }
        post[v] = ind;
    }
    template <class Graph>
    ThreeEdgeCC(const Graph &G)
        : V(G.size()), ind(-1), id(V, -1), pre(V, -1),
          post(V), low(V, INT_MAX), deg(V, 0), path(V, -1),
          uf(V) {
        for (int v = 0; v < V; v++)
            if (pre[v] == -1) dfs(G, v, -1);
        components.reserve(uf.cnt);
        for (int v = 0; v < V; v++)
            if (uf.find(v) == v) {
                id[v] = components.size();
                components.emplace_back(1, v);
                components.back().reserve(uf.getSize(v));
            }
        for (int v = 0; v < V; v++)
            if (id[v] == -1)
                components[id[v]] = id[uf.find(v)].push_back(v);
    }
}

```

3.7. Centroid Decomposition

```

1 void get_center(int now) {
    v[now] = true;
    vtx.push_back(now);
    sz[now] = 1;
    mx[now] = 0;
    for (int u : G[now])
        if (!v[u]) {
            get_center(u);
            mx[now] = max(mx[now], sz[u]);
            sz[now] += sz[u];
        }
}
13 void get_dis(int now, int d, int len) {
    dis[d][now] = cnt;
    v[now] = true;
    for (auto u : G[now])
        if (!v[u.first]) { get_dis(u, d, len + u.second); }
}
19 void dfs(int now, int fa, int d) {
    get_center(now);
    int c = -1;
    for (int i : vtx) {
        if (max(mx[i], (int)vtx.size() - sz[i]) <=
            (int)vtx.size() / 2)
            c = i;
    }
}

```

```

27     v[i] = false;
28 }
29 get_dis(c, d, 0);
30 for (int i : vtx) v[i] = false;
31 v[c] = true;
32 vtx.clear();
33 dep[c] = d;
34 p[c] = fa;
35 for (auto u : G[c])
36     if (u.first != fa && !v[u.first]) {
37         dfs(u.first, c, d + 1);
38     }
39 }
```

3.8. Minimum Mean Cycle

```

1
3 // d[i][j] == 0 if {i,j} !in E
4 long long d[1003][1003], dp[1003][1003];
5
6 pair<long long, long long> MMWC() {
7     memset(dp, 0x3f, sizeof(dp));
8     for (int i = 1; i <= n; ++i) dp[0][i] = 0;
9     for (int i = 1; i <= n; ++i) {
10         for (int j = 1; j <= n; ++j) {
11             for (int k = 1; k <= n; ++k) {
12                 dp[i][k] = min(dp[i - 1][j] + d[j][k], dp[i][k]);
13             }
14         }
15     }
16     long long au = 1ll << 31, ad = 1;
17     for (int i = 1; i <= n; ++i) {
18         if (dp[n][i] == 0x3f3f3f3f3f3f3f3f) continue;
19         long long u = 0, d = 1;
20         for (int j = n - 1; j >= 0; --j) {
21             if ((dp[n][i] - dp[j][i]) * d > u * (n - j)) {
22                 u = dp[n][i] - dp[j][i];
23                 d = n - j;
24             }
25         }
26         if (u * ad < au * d) au = u, ad = d;
27     }
28     long long g = __gcd(au, ad);
29     return make_pair(au / g, ad / g);
30 }
```

3.9. Directed MST

```

1 template <typename T> struct DMST {
2     T g[maxn][maxn], fw[maxn];
3     int n, fr[maxn];
4     bool vis[maxn], inc[maxn];
5     void clear() {
6         for (int i = 0; i < maxn; ++i) {
7             for (int j = 0; j < maxn; ++j) g[i][j] = inf;
8             vis[i] = inc[i] = false;
9         }
10    }
11    void addedge(int u, int v, T w) {
12        g[u][v] = min(g[u][v], w);
13    }
14    T operator()(int root, int _n) {
15        n = _n;
16        if (dfs(root) != n) return -1;
17        T ans = 0;
18        while (true) {
19            for (int i = 1; i <= n; ++i) fw[i] = inf, fr[i] = i;
20            for (int i = 1; i <= n; ++i)
21                if (!inc[i]) {
22                    for (int j = 1; j <= n; ++j) {
23                        if (!inc[j] && i != j && g[j][i] < fw[i]) {
24                            fw[i] = g[j][i];
25                            fr[i] = j;
26                        }
27                    }
28                }
29                int x = -1;
30                for (int i = 1; i <= n; ++i)
31                    if (i != root && !inc[i]) {
32                        int j = i, c = 0;
33                        while (j != root && fr[j] != i && c <= n)
34                            ++c, j = fr[j];
35                        if (j == root || c > n) continue;
36                        else {
37                            x = i;
38                            break;
39                        }
40                    }
41                    if (!~x) {
42                        for (int i = 1; i <= n; ++i)
43                            if (i != root && !inc[i]) ans += fw[i];
44                }
45        }
46    }
47 }
```

```

45     }
46     int y = x;
47     for (int i = 1; i <= n; ++i) vis[i] = false;
48     do {
49         ans += fw[y];
50         y = fr[y];
51         vis[y] = inc[y] = true;
52     } while (y != x);
53     inc[x] = false;
54     for (int k = 1; k <= n; ++k)
55         if (vis[k]) {
56             for (int j = 1; j <= n; ++j)
57                 if (!vis[j]) {
58                     if (g[x][j] > g[k][j]) g[x][j] = g[k][j];
59                     if (g[j][k] < inf &&
60                         g[j][k] - fw[k] < g[j][x])
61                         g[j][x] = g[j][k] - fw[k];
62                 }
63             }
64     }
65     return ans;
66 }
67 int dfs(int now) {
68     int r = 1;
69     vis[now] = true;
70     for (int i = 1; i <= n; ++i)
71         if (g[now][i] < inf && !vis[i]) r += dfs(i);
72     }
73 }
```

3.10. Maximum Clique

```

1 // source: KACTL
2
3 typedef vector<bitset<200>> vb;
4 struct Maxclique {
5     double limit = 0.025, pk = 0;
6     struct Vertex {
7         int i, d = 0;
8     };
9     typedef vector<Vertex> vv;
10    vb e;
11    vv V;
12    vector<vi> C;
13    vi qmax, q, S, old;
14    void init(vv &r) {
15        for (auto &v : r) v.d = 0;
16        for (auto &v : r)
17            for (auto j : r) v.d += e[v.i][j.i];
18        sort(all(r), [] (auto a, auto b) { return a.d > b.d; });
19        int mxD = r[0].d;
20        rep(i, 0, sz(r)) r[i].d = min(i, mxD) + 1;
21    }
22    void expand(vv &R, int lev = 1) {
23        S[lev] += S[lev - 1] - old[lev];
24        old[lev] = S[lev - 1];
25        while (sz(R)) {
26            if (sz(q) + R.back().d <= sz(qmax)) return;
27            q.push_back(R.back().i);
28            vv T;
29            for (auto v : R)
30                if (e[R.back().i][v.i]) T.push_back({v.i});
31            if (S[lev]) {
32                if (S[lev]++ / ++pk < limit) init(T);
33                int j = 0, mxk = 1,
34                    mnk = max(sz(qmax) - sz(q) + 1, 1);
35                C[1].clear(), C[2].clear();
36                for (auto v : T) {
37                    int k = 1;
38                    auto f = [&] (int i) { return e[v.i][i]; };
39                    while (any_of(all(C[k]), f)) k++;
40                    if (k > mxk) mxk = k, C[mxk + 1].clear();
41                    if (k < mnk) T[j++].i = v.i;
42                    C[k].push_back(v.i);
43                }
44                if (j > 0) T[j - 1].d = 0;
45                rep(k, mnk, mxk + 1) for (int i : C[k]) T[j].i = i,
46                                         T[j++].d = k;
47            }
48            expand(T, lev + 1);
49        } else if (sz(q) > sz(qmax)) qmax = q;
50        q.pop_back(), R.pop_back();
51    }
52    vi maxClique() {
53        init(V), expand(V);
54        return qmax;
55    }
56    Maxclique(vb conn)
57        : e(conn), C(sz(e) + 1), S(sz(C)), old(S) {
58            rep(i, 0, sz(e)) V.push_back({i});
59        }
60    };
61 }
```

3.11. Dominator Tree

```

1 // idom[n] is the unique node that strictly dominates n but
2 // does not strictly dominate any other node that strictly
3 // dominates n. idom[n] = 0 if n is entry or the entry
4 // cannot reach n.
5 struct DominatorTree {
6     static const int MAXN = 200010;
7     int n, s;
8     vector<int> g[MAXN], pred[MAXN];
9     vector<int> cov[MAXN];
10    int dfn[MAXN], nfd[MAXN], ts;
11    int par[MAXN];
12    int sdom[MAXN], idom[MAXN];
13    int mom[MAXN], mn[MAXN];
14
15    inline bool cmp(int u, int v) { return dfn[u] < dfn[v]; }
16
17    int eval(int u) {
18        if (mom[u] == u) return u;
19        int res = eval(mom[u]);
20        if (cmp(sdom[mn[mom[u]]], sdom[mn[u]]))
21            mn[u] = mn[mom[u]];
22        return mom[u] = res;
23    }
24
25    void init(int _n, int _s) {
26        n = _n;
27        s = _s;
28        REP1(i, 1, n) {
29            g[i].clear();
30            pred[i].clear();
31            idom[i] = 0;
32        }
33    void add_edge(int u, int v) {
34        g[u].push_back(v);
35        pred[v].push_back(u);
36    }
37    void DFS(int u) {
38        ts++;
39        dfn[u] = ts;
40        nfd[ts] = u;
41        for (int v : g[u])
42            if (dfn[v] == 0) {
43                par[v] = u;
44                DFS(v);
45            }
46    }
47    void build() {
48        ts = 0;
49        REP1(i, 1, n) {
50            dfn[i] = nfd[i] = 0;
51            cov[i].clear();
52            mom[i] = mn[i] = sdom[i] = i;
53        }
54        DFS(s);
55        for (int i = ts; i >= 2; i--) {
56            int u = nfd[i];
57            if (u == 0) continue;
58            for (int v : pred[u])
59                if (dfn[v]) {
60                    eval(v);
61                    if (cmp(sdom[mn[v]], sdom[u]))
62                        sdom[u] = sdom[mn[v]];
63                }
64            cov[sdom[u]].push_back(u);
65            mom[u] = par[u];
66            for (int w : cov[par[u]]) {
67                eval(w);
68                if (cmp(sdom[mn[w]], par[u])) idom[w] = mn[w];
69                else idom[w] = par[u];
70            }
71            cov[par[u]].clear();
72        }
73        REP1(i, 2, ts) {
74            int u = nfd[i];
75            if (u == 0) continue;
76            if (idom[u] != sdom[u]) idom[u] = idom[idom[u]];
77        }
78    }
79 } dom;

```

3.12. Manhattan Distance MST

```

1
2 // returns [(dist, from, to), ...]
3 // then do normal mst afterwards
4 typedef Point<int> P;
5 vector<array<int, 3>> manhattanMST(vector<P> ps) {
6     vi id(sz(ps));
7     iota(all(id), 0);
8     vector<array<int, 3>> edges;

```

```

11    rep(k, 0, 4) {
12        sort(all(id), [&](int i, int j) {
13            return (ps[i] - ps[j]).x < (ps[j] - ps[i]).y;
14        });
15        map<int, int> sweep;
16        for (int i : id) {
17            for (auto it = sweep.lower_bound(-ps[i].y);
18                 it != sweep.end(); sweep.erase(it++)) {
19                int j = it->second;
20                P d = ps[i] - ps[j];
21                if (d.y > d.x) break;
22                edges.push_back({d.y + d.x, i, j});
23            }
24            sweep[-ps[i].y] = i;
25        }
26        for (P &p : ps)
27            if (k & 1) p.x = -p.x;
28            else swap(p.x, p.y);
29    }
30    return edges;
31 }

```

4. Math

4.1. Number Theory

4.1.1. Mod Struct

A list of safe primes: 26003, 27767, 28319, 28979, 29243, 29759, 30467
910927547, 919012223, 947326223, 990669467, 1007939579, 1019126699
929760389146037459, 975500632317046523, 989312547895528379

NTT prime p	$p - 1$	primitive root
65537	$1 \ll 16$	3
998244353	$119 \ll 23$	3
2748779069441	$5 \ll 39$	3
1945555039024054273	$27 \ll 56$	5

Requires: Extended GCD

```

1
3 template <typename T> struct M {
4     static T MOD; // change to constexpr if already known
5     T v;
6     M(T x = 0) {
7         v = (-MOD <= x && x < MOD) ? x : x % MOD;
8         if (v < 0) v += MOD;
9     }
10    explicit operator T() const { return v; }
11    bool operator==(const M &b) const { return v == b.v; }
12    bool operator!=(const M &b) const { return v != b.v; }
13    M operator-() const { return M(-v); }
14    M operator+(M b) { return M(v + b.v); }
15    M operator-(M b) { return M(v - b.v); }
16    M operator*(M b) { return M((__int128)v * b.v % MOD); }
17    M operator/(M b) { return *this * (b ^ (MOD - 2)); }
18    // change above implementation to this if MOD is not prime
19    M inv() {
20        auto [p, _, g] = extgcd(v, MOD);
21        return assert(g == 1), p;
22    }
23    friend M operator^(M a, ll b) {
24        M ans(1);
25        for (; b; b >= 1, a *= a)
26            if (b & 1) ans *= a;
27        return ans;
28    }
29    friend M &operator+=(M &a, M b) { return a = a + b; }
30    friend M &operator-=(M &a, M b) { return a = a - b; }
31    friend M &operator*=(M &a, M b) { return a = a * b; }
32    friend M &operator/=(M &a, M b) { return a = a / b; }
33 }
34 using Mod = M<int>;
35 template <> int Mod::MOD = 1'000'000'007;
36 int &MOD = Mod::MOD;

```

4.1.2. Miller-Rabin

Requires: Mod Struct

```

1
3 // checks if Mod::MOD is prime
4 bool is_prime() {
5     if (MOD < 2 || MOD % 2 == 0) return MOD == 2;
6     Mod A[] = {2, 7, 61}; // for int values (< 2^31)
7     // ll: 2, 325, 9375, 28178, 450775, 9780504, 1795265022
8     int s = __builtin_ctzll(MOD - 1), i;
9     for (Mod a : A) {
10         Mod x = a ^ (MOD >> s);
11         for (i = 0; i < s && (x + 1).v > 2; i++) x *= x;
12         if (i && x != -1) return 0;
13     }
14     return 1;
15 }

```

4.1.3. Linear Sieve

```

1 constexpr ll MAXN = 1000000;
2 bitset<MAXN> is_prime;
3 vector<ll> primes;
4 ll mpf[MAXN], phi[MAXN], mu[MAXN];
5
6 void sieve() {
7     is_prime.set();
8     is_prime[1] = 0;
9     mu[1] = phi[1] = 1;
10    for (ll i = 2; i < MAXN; i++) {
11        if (is_prime[i]) {
12            mpf[i] = i;
13            primes.push_back(i);
14            phi[i] = i - 1;
15            mu[i] = -1;
16        }
17        for (ll p : primes) {
18            if (p > mpf[i] || i * p >= MAXN) break;
19            is_prime[i * p] = 0;
20            mpf[i * p] = p;
21            mu[i * p] = -mu[i];
22            if (i % p == 0)
23                phi[i * p] = phi[i] * p, mu[i * p] = 0;
24            else phi[i * p] = phi[i] * (p - 1);
25        }
26    }
27 }
```

4.1.4. Get Factors

Requires: Linear Sieve

```

1
2 vector<ll> all_factors(ll n) {
3     vector<ll> fac = {1};
4     while (n > 1) {
5         const ll p = mpf[n];
6         vector<ll> cur = {1};
7         while (n % p == 0) {
8             n /= p;
9             cur.push_back(cur.back() * p);
10        }
11        vector<ll> tmp;
12        for (auto x : fac)
13            for (auto y : cur) tmp.push_back(x * y);
14        tmp.swap(fac);
15    }
16    return fac;
17 }
```

4.1.5. Binary GCD

```

1 // returns the gcd of non-negative a, b
2 ull bin_gcd(ull a, ull b) {
3     if (!a || !b) return a + b;
4     int s = __builtin_ctzll(a | b);
5     a >>= __builtin_ctzll(a);
6     while (b) {
7         if ((b >>= __builtin_ctzll(b)) < a) swap(a, b);
8         b -= a;
9     }
10    return a << s;
11 }
```

4.1.6. Extended GCD

```

1 // returns (p, q, g): p * a + q * b == g == gcd(a, b)
2 // g is not guaranteed to be positive when a < 0 or b < 0
3 tuple<ll, ll, ll> extgcd(ll a, ll b) {
4     ll s = 1, t = 0, u = 0, v = 1;
5     while (b) {
6         ll q = a / b;
7         swap(a -= q * b, b);
8         swap(s -= q * t, t);
9         swap(u -= q * v, v);
10    }
11    return {s, u, a};
12 }
```

4.1.7. Chinese Remainder Theorem

Requires: Extended GCD

```

1 // for 0 <= a < m, 0 <= b < n, returns the smallest x >= 0
2 // such that x % m == a and x % n == b
3 ll crt(ll a, ll m, ll b, ll n) {
4     if (n > m) swap(a, b), swap(m, n);
5     auto [x, y, g] = extgcd(m, n);
6     assert((a - b) % g == 0); // no solution
7     x = ((b - a) / g * x) % (n / g) * m + a;
8     return x < 0 ? x + m / g * n : x;
9 }
```

4.1.8. Baby-Step Giant-Step

Requires: Mod Struct

```

1
2 // returns x such that a ^ x = b where x \in [l, r]
3 ll bsgs(Mod a, Mod b, ll l = 0, ll r = MOD - 1) {
4     int m = sqrt(r - l) + 1, i;
5     unordered_map<ll, ll> tb;
6     Mod d = (a ^ l) / b;
7     for (i = 0, d = (a ^ l) / b; i < m; i++, d *= a)
8         if (d == 1) return l + i;
9         else tb[(ll)d] = l + i;
10    Mod c = Mod(1) / (a ^ m);
11    for (i = 0, d = 1; i < m; i++, d *= a)
12        if (auto j = tb.find((ll)d); j != tb.end())
13            return j->second + i * m;
14    return assert(0), -1; // no solution
15 }
```

4.1.9. Pollard's Rho

```

1 ll f(ll x, ll mod) { return (x * x + 1) % mod; }
2 // n should be composite
3 ll pollard_rho(ll n) {
4     if (!(n & 1)) return 2;
5     while (1) {
6         ll y = 2, x = RNG() % (n - 1) + 1, res = 1;
7         for (int sz = 2; res == 1; sz *= 2) {
8             for (int i = 0; i < sz && res <= 1; i++) {
9                 x = f(x, n);
10                res = __gcd(abs(x - y), n);
11            }
12            y = x;
13        }
14        if (res != 0 && res != n) return res;
15    }
16 }
```

4.1.10. Tonelli-Shanks Algorithm

Requires: Mod Struct

```

1
2 int legendre(Mod a) {
3     if (a == 0) return 0;
4     return (a ^ ((MOD - 1) / 2)) == 1 ? 1 : -1;
5 }
6 Mod sqrt(Mod a) {
7     assert(legendre(a) != -1); // no solution
8     ll p = MOD, s = p - 1;
9     if (a == 0) return 0;
10    if (p == 2) return 1;
11    if (p % 4 == 3) return a ^ ((p + 1) / 4);
12    int r, m;
13    for (r = 0; !(s & 1); r++) s >>= 1;
14    Mod n = 2;
15    while (legendre(n) != -1) n += 1;
16    Mod x = a ^ ((s + 1) / 2), b = a ^ s, g = n ^ s;
17    while (b != 1) {
18        Mod t = b;
19        for (m = 0; t != 1; m++) t *= t;
20        Mod gs = g ^ (1LL << (r - m - 1));
21        g = gs * gs, x *= gs, b *= g, r = m;
22    }
23    return x;
24 }
25 // to get sqrt(X) modulo p^k, where p is an odd prime:
26 // c = x^2 (mod p), c = X^2 (mod p^k), q = p^(k-1)
27 // X = x^q * c^{((p^k-2q+1)/2)} (mod p^k)
28 }
```

4.1.11. Chinese Sieve

```

1 const ll N = 1000000;
2 // f, g, h multiplicative, h = f (dirichlet convolution) g
3 ll pre_g(ll n);
4 ll pre_h(ll n);
5 // preprocessed prefix sum of f
6 ll pre_f[N];
7 // prefix sum of multiplicative function f
8 ll solve_f(ll n) {
9     static unordered_map<ll, ll> m;
10    if (n < N) return pre_f[n];
11    if (m.count(n)) return m[n];
12    ll ans = pre_h(n);
13    for (ll l = 2, r; l <= n; l = r + 1) {
14        r = n / (n / l);
15        ans -= (pre_g(r) - pre_g(l - 1)) * djs_f(n / l);
16    }
17    return m[n] = ans;
18 }
```

4.1.12. Rational Number Binary Search

```

1 struct QQ {
2     ll p, q;
3     QQ go(QQ b, ll d) { return {p + b.p * d, q + b.q * d}; }
4 };
5 bool pred(QQ);
6 // returns smallest p/q in [lo, hi] such that
7 // pred(p/q) is true, and 0 <= p, q <= N
8 QQ frac_bs(ll N) {
9     QQ lo{0, 1}, hi{1, 0};
10    if (pred(lo)) return lo;
11    assert(pred(hi));
12    bool dir = 1, L = 1, H = 1;
13    for (; L || H; dir = !dir) {
14        ll len = 0, step = 1;
15        for (int t = 0; t < 2 && (t ? step /= 2 : step *= 2);)
16            if (QQ mid = hi.go(lo, len + step);
17                mid.p > N || mid.q > N || dir ^ pred(mid))
18                t++;
19            else len += step;
20            swap(lo, hi = hi.go(lo, len));
21            (dir ? L : H) = !!len;
22    }
23    return dir ? hi : lo;
}

```

4.1.13. Farey Sequence

```

1 // returns (e/f), where (a/b, c/d, e/f) are
2 // three consecutive terms in the order n farey sequence
3 // to start, call next_farey(n, 0, 1, 1, n)
4 pll next_farey(ll n, ll a, ll b, ll c, ll d) {
5     ll p = (n + b) / d;
6     return pll(p * c - a, p * d - b);
7 }

```

4.2. Combinatorics

4.2.1. Matroid Intersection

This template assumes 2 weighted matroids of the same type, and that removing an element is much more expensive than checking if one can be added. **Remember to change the implementation details.**

The ground set is $0, 1, \dots, n - 1$, where element i has weight $w[i]$. For the unweighted version, remove weights and change BF/SPFA to BFS.

```

1 constexpr int N = 100;
2 constexpr int INF = 1e9;
3
4 struct Matroid {           // represents an independent set
5     Matroid(bitset<N>);   // initialize from an independent set
6     bool can_add(int);     // if adding will break independence
7     Matroid remove(int);   // removing from the set
8 };
9
10 auto matroid_intersection(int n, const vector<int> &w) {
11     bitset<N> S;
12     for (int sz = 1; sz <= n; sz++) {
13         Matroid M1(S), M2(S);
14
15         vector<vector<pii>> e(n + 2);
16         for (int j = 0; j < n; j++)
17             if (!S[j]) {
18                 if (M1.can_add(j)) e[n].emplace_back(j, -w[j]);
19                 if (M2.can_add(j)) e[j].emplace_back(n + 1, 0);
20             }
21         for (int i = 0; i < n; i++)
22             if (S[i]) {
23                 Matroid T1 = M1.remove(i), T2 = M2.remove(i);
24                 for (int j = 0; j < n; j++)
25                     if (!S[j]) {
26                         if (T1.can_add(j)) e[i].emplace_back(j, -w[j]);
27                         if (T2.can_add(j)) e[j].emplace_back(i, w[i]);
28                     }
29
30         vector<pii> dis(n + 2, {INF, 0});
31         vector<int> prev(n + 2, -1);
32         dis[0] = {0, 0};
33         // change to SPFA for more speed, if necessary
34         bool upd = 1;
35         while (upd) {
36             upd = 0;
37             for (int u = 0; u < n + 2; u++)
38                 for (auto [v, c] : e[u]) {
39                     pii x(dis[u].first + c, dis[u].second + 1);
40                     if (x < dis[v]) dis[v] = x, prev[v] = u, upd = 1;
41                 }
42
43         if (dis[n + 1].first < INF)
44             for (int x = prev[n + 1]; x != n; x = prev[x])

```

```

47         S.flip(x);
48     else break;
49
50     // S is the max-weighted independent set with size sz
51 } return S;
52 }

```

4.2.2. De Bruijn Sequence

```

1 int res[kN], aux[kN], a[kN], sz;
2 void Rec(int t, int p, int n, int k) {
3     if (t > n) {
4         if (n % p == 0)
5             for (int i = 1; i <= p; ++i) res[sz++] = aux[i];
6     } else {
7         aux[t] = aux[t - p];
8         Rec(t + 1, p, n, k);
9         for (aux[t] = aux[t - p] + 1; aux[t] < k; ++aux[t])
10            Rec(t + 1, t, n, k);
11    }
12 }
13 int DeBruijn(int k, int n) {
14     // return cyclic string of length k^n such that every
15     // string of length n using k character appears as a
16     // substring.
17     if (k == 1) return res[0] = 0, 1;
18     fill(aux, aux + k * n, 0);
19     return sz = 0, Rec(1, 1, n, k), sz;
20
21 // dd jflkj s fjkj jlk
22 }

```

4.2.3. Multinomial

```

1
2
3 // ways to permute v[i]
4 ll multinomial(vi &v) {
5     ll c = 1, m = v.empty() ? 1 : v[0];
6     for (int i = 1; i < v.size(); i++)
7         for (int j = 0; i < v[i]; j++) c = c * ++m / (j + 1);
8 }

```

4.3. Algebra

4.3.1. Formal Power Series

```

1
2
3 template <typename mint>
4 struct FormalPowerSeries : vector<mint> {
5     using vector<mint>::vector;
6     using FPS = FormalPowerSeries;
7
8     FPS &operator+=(const FPS &r) {
9         if (r.size() > this->size()) this->resize(r.size());
10        for (int i = 0; i < (int)r.size(); i++)
11            (*this)[i] += r[i];
12        return *this;
13    }
14
15     FPS &operator+=(const mint &r) {
16         if (this->empty()) this->resize(1);
17         (*this)[0] += r;
18         return *this;
19    }
20
21     FPS &operator-=(const FPS &r) {
22         if (r.size() > this->size()) this->resize(r.size());
23         for (int i = 0; i < (int)r.size(); i++)
24             (*this)[i] -= r[i];
25         return *this;
26    }
27
28     FPS &operator-=(const mint &r) {
29         if (this->empty()) this->resize(1);
30         (*this)[0] -= r;
31         return *this;
32    }
33
34     FPS &operator*=(const mint &v) {
35         for (int k = 0; k < (int)this->size(); k++)
36             (*this)[k] *= v;
37         return *this;
38    }
39
40     FPS &operator/=(const FPS &r) {
41         if (this->size() < r.size()) {
42             this->clear();
43             return *this;
44         }
45
46         mint inv = 1 / r[0];
47         for (int i = 0; i < this->size(); i++)
48             this->operator[](i) *= inv;
49
50         this->normalize();
51
52         if (this->empty())
53             this->clear();
54         else if (this->size() == 1)
55             this->operator[](0) = 1;
56         else if (this->size() == 2)
57             this->operator[](1) = 1;
58         else
59             this->operator[](0) = 1;
60
61         return *this;
62    }
63
64     mint operator[](int i) const {
65         if (i < 0) return 0;
66         if (i > this->size() - 1) return 0;
67         return this->operator[](i);
68    }
69
70     mint operator[](int i) {
71         if (i < 0) return 0;
72         if (i > this->size() - 1) return 0;
73         return this->operator[](i);
74    }
75
76     mint operator[](int i) const {
77         if (i < 0) return 0;
78         if (i > this->size() - 1) return 0;
79         return this->operator[](i);
80    }
81
82     mint operator[](int i) {
83         if (i < 0) return 0;
84         if (i > this->size() - 1) return 0;
85         return this->operator[](i);
86    }
87
88     mint operator[](int i) const {
89         if (i < 0) return 0;
90         if (i > this->size() - 1) return 0;
91         return this->operator[](i);
92    }
93
94     mint operator[](int i) {
95         if (i < 0) return 0;
96         if (i > this->size() - 1) return 0;
97         return this->operator[](i);
98    }
99
100    mint operator[](int i) const {
101        if (i < 0) return 0;
102        if (i > this->size() - 1) return 0;
103        return this->operator[](i);
104    }
105
106    mint operator[](int i) {
107        if (i < 0) return 0;
108        if (i > this->size() - 1) return 0;
109        return this->operator[](i);
110    }
111
112    mint operator[](int i) const {
113        if (i < 0) return 0;
114        if (i > this->size() - 1) return 0;
115        return this->operator[](i);
116    }
117
118    mint operator[](int i) {
119        if (i < 0) return 0;
120        if (i > this->size() - 1) return 0;
121        return this->operator[](i);
122    }
123
124    mint operator[](int i) const {
125        if (i < 0) return 0;
126        if (i > this->size() - 1) return 0;
127        return this->operator[](i);
128    }
129
130    mint operator[](int i) {
131        if (i < 0) return 0;
132        if (i > this->size() - 1) return 0;
133        return this->operator[](i);
134    }
135
136    mint operator[](int i) const {
137        if (i < 0) return 0;
138        if (i > this->size() - 1) return 0;
139        return this->operator[](i);
140    }
141
142    mint operator[](int i) {
143        if (i < 0) return 0;
144        if (i > this->size() - 1) return 0;
145        return this->operator[](i);
146    }
147
148    mint operator[](int i) const {
149        if (i < 0) return 0;
150        if (i > this->size() - 1) return 0;
151        return this->operator[](i);
152    }
153
154    mint operator[](int i) {
155        if (i < 0) return 0;
156        if (i > this->size() - 1) return 0;
157        return this->operator[](i);
158    }
159
160    mint operator[](int i) const {
161        if (i < 0) return 0;
162        if (i > this->size() - 1) return 0;
163        return this->operator[](i);
164    }
165
166    mint operator[](int i) {
167        if (i < 0) return 0;
168        if (i > this->size() - 1) return 0;
169        return this->operator[](i);
170    }
171
172    mint operator[](int i) const {
173        if (i < 0) return 0;
174        if (i > this->size() - 1) return 0;
175        return this->operator[](i);
176    }
177
178    mint operator[](int i) {
179        if (i < 0) return 0;
180        if (i > this->size() - 1) return 0;
181        return this->operator[](i);
182    }
183
184    mint operator[](int i) const {
185        if (i < 0) return 0;
186        if (i > this->size() - 1) return 0;
187        return this->operator[](i);
188    }
189
190    mint operator[](int i) {
191        if (i < 0) return 0;
192        if (i > this->size() - 1) return 0;
193        return this->operator[](i);
194    }
195
196    mint operator[](int i) const {
197        if (i < 0) return 0;
198        if (i > this->size() - 1) return 0;
199        return this->operator[](i);
200    }
201
202    mint operator[](int i) {
203        if (i < 0) return 0;
204        if (i > this->size() - 1) return 0;
205        return this->operator[](i);
206    }
207
208    mint operator[](int i) const {
209        if (i < 0) return 0;
210        if (i > this->size() - 1) return 0;
211        return this->operator[](i);
212    }
213
214    mint operator[](int i) {
215        if (i < 0) return 0;
216        if (i > this->size() - 1) return 0;
217        return this->operator[](i);
218    }
219
220    mint operator[](int i) const {
221        if (i < 0) return 0;
222        if (i > this->size() - 1) return 0;
223        return this->operator[](i);
224    }
225
226    mint operator[](int i) {
227        if (i < 0) return 0;
228        if (i > this->size() - 1) return 0;
229        return this->operator[](i);
230    }
231
232    mint operator[](int i) const {
233        if (i < 0) return 0;
234        if (i > this->size() - 1) return 0;
235        return this->operator[](i);
236    }
237
238    mint operator[](int i) {
239        if (i < 0) return 0;
240        if (i > this->size() - 1) return 0;
241        return this->operator[](i);
242    }
243
244    mint operator[](int i) const {
245        if (i < 0) return 0;
246        if (i > this->size() - 1) return 0;
247        return this->operator[](i);
248    }
249
250    mint operator[](int i) {
251        if (i < 0) return 0;
252        if (i > this->size() - 1) return 0;
253        return this->operator[](i);
254    }
255
256    mint operator[](int i) const {
257        if (i < 0) return 0;
258        if (i > this->size() - 1) return 0;
259        return this->operator[](i);
260    }
261
262    mint operator[](int i) {
263        if (i < 0) return 0;
264        if (i > this->size() - 1) return 0;
265        return this->operator[](i);
266    }
267
268    mint operator[](int i) const {
269        if (i < 0) return 0;
270        if (i > this->size() - 1) return 0;
271        return this->operator[](i);
272    }
273
274    mint operator[](int i) {
275        if (i < 0) return 0;
276        if (i > this->size() - 1) return 0;
277        return this->operator[](i);
278    }
279
280    mint operator[](int i) const {
281        if (i < 0) return 0;
282        if (i > this->size() - 1) return 0;
283        return this->operator[](i);
284    }
285
286    mint operator[](int i) {
287        if (i < 0) return 0;
288        if (i > this->size() - 1) return 0;
289        return this->operator[](i);
290    }
291
292    mint operator[](int i) const {
293        if (i < 0) return 0;
294        if (i > this->size() - 1) return 0;
295        return this->operator[](i);
296    }
297
298    mint operator[](int i) {
299        if (i < 0) return 0;
300        if (i > this->size() - 1) return 0;
301        return this->operator[](i);
302    }
303
304    mint operator[](int i) const {
305        if (i < 0) return 0;
306        if (i > this->size() - 1) return 0;
307        return this->operator[](i);
308    }
309
310    mint operator[](int i) {
311        if (i < 0) return 0;
312        if (i > this->size() - 1) return 0;
313        return this->operator[](i);
314    }
315
316    mint operator[](int i) const {
317        if (i < 0) return 0;
318        if (i > this->size() - 1) return 0;
319        return this->operator[](i);
320    }
321
322    mint operator[](int i) {
323        if (i < 0) return 0;
324        if (i > this->size() - 1) return 0;
325        return this->operator[](i);
326    }
327
328    mint operator[](int i) const {
329        if (i < 0) return 0;
330        if (i > this->size() - 1) return 0;
331        return this->operator[](i);
332    }
333
334    mint operator[](int i) {
335        if (i < 0) return 0;
336        if (i > this->size() - 1) return 0;
337        return this->operator[](i);
338    }
339
340    mint operator[](int i) const {
341        if (i < 0) return 0;
342        if (i > this->size() - 1) return 0;
343        return this->operator[](i);
344    }
345
346    mint operator[](int i) {
347        if (i < 0) return 0;
348        if (i > this->size() - 1) return 0;
349        return this->operator[](i);
350    }
351
352    mint operator[](int i) const {
353        if (i < 0) return 0;
354        if (i > this->size() - 1) return 0;
355        return this->operator[](i);
356    }
357
358    mint operator[](int i) {
359        if (i < 0) return 0;
360        if (i > this->size() - 1) return 0;
361        return this->operator[](i);
362    }
363
364    mint operator[](int i) const {
365        if (i < 0) return 0;
366        if (i > this->size() - 1) return 0;
367        return this->operator[](i);
368    }
369
370    mint operator[](int i) {
371        if (i < 0) return 0;
372        if (i > this->size() - 1) return 0;
373        return this->operator[](i);
374    }
375
376    mint operator[](int i) const {
377        if (i < 0) return 0;
378        if (i > this->size() - 1) return 0;
379        return this->operator[](i);
380    }
381
382    mint operator[](int i) {
383        if (i < 0) return 0;
384        if (i > this->size() - 1) return 0;
385        return this->operator[](i);
386    }
387
388    mint operator[](int i) const {
389        if (i < 0) return 0;
390        if (i > this->size() - 1) return 0;
391        return this->operator[](i);
392    }
393
394    mint operator[](int i) {
395        if (i < 0) return 0;
396        if (i > this->size() - 1) return 0;
397        return this->operator[](i);
398    }
399
400    mint operator[](int i) const {
401        if (i < 0) return 0;
402        if (i > this->size() - 1) return 0;
403        return this->operator[](i);
404    }
405
406    mint operator[](int i) {
407        if (i < 0) return 0;
408        if (i > this->size() - 1) return 0;
409        return this->operator[](i);
410    }
411
412    mint operator[](int i) const {
413        if (i < 0) return 0;
414        if (i > this->size() - 1) return 0;
415        return this->operator[](i);
416    }
417
418    mint operator[](int i) {
419        if (i < 0) return 0;
420        if (i > this->size() - 1) return 0;
421        return this->operator[](i);
422    }
423
424    mint operator[](int i) const {
425        if (i < 0) return 0;
426        if (i > this->size() - 1) return 0;
427        return this->operator[](i);
428    }
429
430    mint operator[](int i) {
431        if (i < 0) return 0;
432        if (i > this->size() - 1) return 0;
433        return this->operator[](i);
434    }
435
436    mint operator[](int i) const {
437        if (i < 0) return 0;
438        if (i > this->size() - 1) return 0;
439        return this->operator[](i);
440    }
441
442    mint operator[](int i) {
443        if (i < 0) return 0;
444        if (i > this->size() - 1) return 0;
445        return this->operator[](i);
446    }
447
448    mint operator[](int i) const {
449        if (i < 0) return 0;
450        if (i > this->size() - 1) return 0;
451        return this->operator[](i);
452    }
453
454    mint operator[](int i) {
455        if (i < 0) return 0;
456        if (i > this->size() - 1) return 0;
457        return this->operator[](i);
458    }
459
460    mint operator[](int i) const {
461        if (i < 0) return 0;
462        if (i > this->size() - 1) return 0;
463        return this->operator[](i);
464    }
465
466    mint operator[](int i) {
467        if (i < 0) return 0;
468        if (i > this->size() - 1) return 0;
469        return this->operator[](i);
470    }
471
472    mint operator[](int i) const {
473        if (i < 0) return 0;
474        if (i > this->size() - 1) return 0;
475        return this->operator[](i);
476    }
477
478    mint operator[](int i) {
479        if (i < 0) return 0;
480        if (i > this->size() - 1) return 0;
481        return this->operator[](i);
482    }
483
484    mint operator[](int i) const {
485        if (i < 0) return 0;
486        if (i > this->size() - 1) return 0;
487        return this->operator[](i);
488    }
489
490    mint operator[](int i) {
491        if (i < 0) return 0;
492        if (i > this->size() - 1) return 0;
493        return this->operator[](i);
494    }
495
496    mint operator[](int i) const {
497        if (i < 0) return 0;
498        if (i > this->size() - 1) return 0;
499        return this->operator[](i);
500    }
501
502    mint operator[](int i) {
503        if (i < 0) return 0;
504        if (i > this->size() - 1) return 0;
505        return this->operator[](i);
506    }
507
508    mint operator[](int i) const {
509        if (i < 0) return 0;
510        if (i > this->size() - 1) return 0;
511        return this->operator[](i);
512    }
513
514    mint operator[](int i) {
515        if (i < 0) return 0;
516        if (i > this->size() - 1) return 0;
517        return this->operator[](i);
518    }
519
520    mint operator[](int i) const {
521        if (i < 0) return 0;
522        if (i > this->size() - 1) return 0;
523        return this->operator[](i);
524    }
525
526    mint operator[](int i) {
527        if (i < 0) return 0;
528        if (i > this->size() - 1) return 0;
529        return this->operator[](i);
530    }
531
532    mint operator[](int i) const {
533        if (i < 0) return 0;
534        if (i > this->size() - 1) return 0;
535        return this->operator[](i);
536    }
537
538    mint operator[](int i) {
539        if (i < 0) return 0;
540        if (i > this->size() - 1) return 0;
541        return this->operator[](i);
542    }
543
544    mint operator[](int i) const {
545        if (i < 0) return 0;
546        if (i > this->size() - 1) return 0;
547        return this->operator[](i);
548    }
549
550    mint operator[](int i) {
551        if (i < 0) return 0;
552        if (i > this->size() - 1) return 0;
553        return this->operator[](i);
554    }
555
556    mint operator[](int i) const {
557        if (i < 0) return 0;
558        if (i > this->size() - 1) return 0;
559        return this->operator[](i);
560    }
561
562    mint operator[](int i) {
563        if (i < 0) return 0;
564        if (i > this->size() - 1) return 0;
565        return this->operator[](i);
566    }
567
568    mint operator[](int i) const {
569        if (i < 0) return 0;
570        if (i > this->size() - 1) return 0;
571        return this->operator[](i);
572    }
573
574    mint operator[](int i) {
575        if (i < 0) return 0;
576        if (i > this->size() - 1) return 0;
577        return this->operator[](i);
578    }
579
580    mint operator[](int i) const {
581        if (i < 0) return 0;
582        if (i > this->size() - 1) return 0;
583        return this->operator[](i);
584    }
585
586    mint operator[](int i) {
587        if (i < 0) return 0;
588        if (i > this->size() - 1) return 0;
589        return this->operator[](i);
590    }
591
592    mint operator[](int i) const {
593        if (i < 0) return 0;
594        if (i > this->size() - 1) return 0;
595        return this->operator[](i);
596    }
597
598    mint operator[](int i) {
599        if (i < 0) return 0;
600        if (i > this->size() - 1) return 0;
601        return this->operator[](i);
602    }
603
604    mint operator[](int i) const {
605        if (i < 0) return 0;
606        if (i > this->size() - 1) return 0;
607        return this->operator[](i);
608    }
609
610    mint operator[](int i) {
611        if (i < 0) return 0;
612        if (i > this->size() - 1) return 0;
613        return this->operator[](i);
614    }
615
616    mint operator[](int i) const {
617        if (i < 0) return 0;
618        if (i > this->size() - 1) return 0;
619        return this->operator[](i);
620    }
621
622    mint operator[](int i) {
623        if (i < 0) return 0;
624        if (i > this->size() - 1) return 0;
625        return this->operator[](i);
626    }
627
628    mint operator[](int i) const {
629        if (i < 0) return 0;
630        if (i > this->size() - 1) return 0;
631        return this->operator[](i);
632    }
633
634    mint operator[](int i) {
635        if (i < 0) return 0;
636        if (i > this->size() - 1) return 0;
637        return this->operator[](i);
638    }
639
640    mint operator[](int i) const {
641        if (i < 0) return 0;
642        if (i > this->size() - 1) return 0;
643        return this->operator[](i);
644    }
645
646    mint operator[](int i) {
647        if (i < 0) return 0;
648        if (i > this->size() - 1) return 0;
649        return this->operator[](i);
650    }
651
652    mint operator[](int i) const {
653        if (i < 0) return 0;
654        if (i > this->size() - 1) return 0;
655        return this->operator[](i);
656    }
657
658    mint operator[](int i) {
659        if (i < 0) return 0;
660        if (i > this->size() - 1) return 0;
661        return this->operator[](i);
662    }
663
664    mint operator[](int i) const {
665        if (i < 0) return 0;
666        if (i > this->size() - 1) return 0;
667        return this->operator[](i);
668    }
669
670    mint operator[](int i) {
671        if (i < 0) return 0;
672        if (i > this->size() - 1) return 0;
673        return this->operator[](i);
674    }
675
676    mint operator[](int i) const {
677        if (i < 0) return 0;
678        if (i > this->size() - 1) return 0;
679        return this->operator[](i);
680    }
681
682    mint operator[](int i) {
683        if (i < 0) return 0;
684        if (i > this->size() - 1) return 0;
685        return this->operator[](i);
686    }
687
688    mint operator[](int i) const {
689        if (i < 0) return 0;
690        if (i > this->size() - 1) return 0;
691        return this->operator[](i);
692    }
693
694    mint operator[](int i) {
695        if (i < 0) return 0;
696        if (i > this->size() - 1) return 0;
697        return this->operator[](i);
698    }
699
700    mint operator[](int i) const {
701        if (i < 0) return 0;
702        if (i > this->size() - 1) return 0;
703        return this->operator[](i);
704    }
705
706    mint operator[](int i) {
707        if (i < 0) return 0;
708        if (i > this->size() - 1) return 0;
709        return this->operator[](i);
710    }
711
712    mint operator[](int i) const {
713        if (i < 0) return 0;
714        if (i > this->size() - 1) return 0;
715        return this->operator[](i);
716    }
717
718    mint operator[](int i) {
719        if (i < 0) return 0;
720        if (i > this->size() - 1) return 0;
721        return this->operator[](i);
722    }
723
724    mint operator[](int i) const {
725        if (i < 0) return 0;
726        if (i > this->size() - 1) return 0;
727        return this->operator[](i);
728    }
72
```

```

45 }
46     int n = this->size() - r.size() + 1;
47     if ((int)r.size() <= 64) {
48         FPS f(*this), g(r);
49         g.shrink();
50         mint coeff = g.back().inverse();
51         for (auto &x : g) x *= coeff;
52         int deg = (intf.size() - (int)g.size() + 1;
53         int gs = g.size();
54         FPS quo(deg);
55         for (int i = deg - 1; i >= 0; i--) {
56             quo[i] = f[i + gs - 1];
57             for (int j = 0; j < gs; j++)
58                 f[i + j] -= quo[i] * g[j];
59         }
60         *this = quo * coeff;
61         this->resize(n, mint(0));
62         return *this;
63     }
64     return *this = ((*this).rev().pre(n) * r.rev().inv(n))
65         .pre(n)
66         .rev();
67 }

68 FPS &operator%=(const FPS &r) {
69     *this -= *this / r * r;
70     shrink();
71     return *this;
72 }

73 FPS operator+(const FPS &r) const {
74     return FPS(*this) += r;
75 }
76 FPS operator+(const mint &v) const {
77     return FPS(*this) += v;
78 }
79 FPS operator-(const FPS &r) const {
80     return FPS(*this) -= r;
81 }
82 FPS operator-(const mint &v) const {
83     return FPS(*this) -= v;
84 }
85 FPS operator*(const FPS &r) const {
86     return FPS(*this) *= r;
87 }
88 FPS operator*(const mint &v) const {
89     return FPS(*this) *= v;
90 }
91 FPS operator/(const FPS &r) const {
92     return FPS(*this) /= r;
93 }
94 FPS operator%(const FPS &r) const {
95     return FPS(*this) %= r;
96 }
97 FPS operator-() const {
98     FPS ret(this->size());
99     for (int i = 0; i < (int)this->size(); i++)
100        ret[i] = -(*this)[i];
101    return ret;
102 }

103 void shrink() {
104     while (this->size() && this->back() == mint(0))
105         this->pop_back();
106 }

107 FPS rev() const {
108     FPS ret(*this);
109     reverse(begin(ret), end(ret));
110     return ret;
111 }

112 FPS dot(FPS r) const {
113     FPS ret(min(this->size(), r.size()));
114     for (int i = 0; i < (int)ret.size(); i++)
115         ret[i] = (*this)[i] * r[i];
116     return ret;
117 }

118 FPS pre(int sz) const {
119     return FPS(begin(*this),
120                begin(*this) + min((int)this->size(), sz));
121 }

122 FPS operator>(<int sz) const {
123     if ((int)this->size() <= sz) return {};
124     FPS ret(*this);
125     ret.erase(ret.begin(), ret.begin() + sz);
126     return ret;
127 }

128 FPS operator<<(<int sz) const {
129     FPS ret(*this);
130     ret.insert(ret.begin(), sz, mint(0));
131 }

```

```

139     return ret;
140 }

141 FPS diff() const {
142     const int n = (int)this->size();
143     FPS ret(max(0, n - 1));
144     mint one(1), coeff(1);
145     for (int i = 1; i < n; i++) {
146         ret[i - 1] = (*this)[i] * coeff;
147         coeff += one;
148     }
149     return ret;
150 }

151 FPS integral() const {
152     const int n = (int)this->size();
153     FPS ret(n + 1);
154     ret[0] = mint(0);
155     if (n > 0) ret[1] = mint(1);
156     auto mod = mint::get_mod();
157     for (int i = 2; i <= n; i++)
158         ret[i] = (-ret[mod % i]) * (mod / i);
159     for (int i = 0; i < n; i++) ret[i + 1] *= (*this)[i];
160     return ret;
161 }

162 mint eval(mint x) const {
163     mint r = 0, w = 1;
164     for (auto &v : *this) r += w * v, w *= x;
165     return r;
166 }

167 FPS log(int deg = -1) const {
168     assert((*this)[0] == mint(1));
169     if (deg == -1) deg = (int)this->size();
170     return (this->diff() * this->inv(deg))
171         .pre(deg - 1)
172         .integral();
173 }

174 FPS pow(int64_t k, int deg = -1) const {
175     const int n = (int)this->size();
176     if (deg == -1) deg = n;
177     for (int i = 0; i < n; i++) {
178         if ((*this)[i] != mint(0)) {
179             if (i * k > deg) return FPS(deg, mint(0));
180             mint rev = mint(1) / (*this)[i];
181             FPS ret =
182                 (((*this * rev) >> i).log(deg) * k).exp(deg) *
183                 ((*this)[i].pow(k));
184             ret = (ret << (i * k)).pre(deg);
185             if ((int)ret.size() < deg) ret.resize(deg, mint(0));
186             return ret;
187         }
188     }
189     return FPS(deg, mint(0));
190 }

191 static void *ntt_ptr;
192 static void set_fft();
193 FPS &operator*=(const FPS &r);
194 void ntt();
195 void intt();
196 void ntt_doubling();
197 static int ntt_pr();
198 FPS inv(int deg = -1) const;
199 FPS exp(int deg = -1) const;
200 };
201 template <typename mint>
202 void *FormalPowerSeries<mint>::ntt_ptr = nullptr;

```

4.4. Theorems

4.4.1. Kirchhoff's Theorem

Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G , where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i, j) in G .

- The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
- The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.

4.4.2. Tutte's Matrix

Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i < j$ and $(i, j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{\text{rank}(D)}{2}$ is the maximum matching on G .

4.4.3. Cayley's Formula

- Given a degree sequence d_1, d_2, \dots, d_n for each labeled vertices, there are

$$\frac{(n-2)!}{(d_1-1)!(d_2-1)! \cdots (d_n-1)!}$$

spanning trees.

- Let $T_{n,k}$ be the number of labeled forests on n vertices with k components, such that vertex $1, 2, \dots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.

4.4.4. Erdős–Gallai Theorem

A sequence of non-negative integers $d_1 \geq d_2 \geq \dots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + d_2 + \dots + d_n$ is even and

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$$

holds for all $1 \leq k \leq n$.

4.4.5. Burnside's Lemma

Let X be a set and G be a group that acts on X . For $g \in G$, denote by X^g the elements fixed by g :

$$X^g = \{x \in X \mid gx \in X\}$$

Then

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$$

5. Numeric

5.1. Barrett Reduction

```
1 using ull = unsigned long long;
2 using ul = uint128_t;
3 // very fast calculation of a % m
4 struct reduction {
5     const ull m, d;
6     explicit reduction(ull m) : m(m), d(((ul)1 << 64) / m) {}
7     inline ull operator()(ull a) const {
8         ull q = (ull)((ul)d * a) >> 64;
9         return (a -= q * m) >= m ? a - m : a;
10    }
11 }
```

5.2. Long Long Multiplication

```
1 using ull = unsigned long long;
2 using ll = long long;
3 using ld = long double;
4 // returns a * b % M where a, b < M < 2**63
5 ull mult(ull a, ull b, ull M) {
6     ll ret = a * b - M * ull(ld(a) * ld(b) / ld(M));
7     return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
```

5.3. Fast Fourier Transform

```
1 template <typename T>
2 void fft_(int n, vector<T> &a, vector<T> &rt, bool inv) {
3     vector<int> br(n);
4     for (int i = 1; i < n; i++) {
5         br[i] = (i & 1) ? br[i - 1] + n / 2 : br[i / 2] / 2;
6         if (br[i] > i) swap(a[i], a[br[i]]);
7     }
8     for (int len = 2; len <= n; len *= 2)
9         for (int i = 0; i < n; i += len)
10            for (int j = 0; j < len / 2; j++) {
11                int pos = n / len * (inv ? len - j : j);
12                T u = a[i + j], v = a[i + j + len / 2] * rt[pos];
13                a[i + j] = u + v, a[i + j + len / 2] = u - v;
14            }
15     if (T minv = T(1) / T(n); inv)
16         for (T &x : a) x *= minv;
17 }
```

Requires: Mod Struct

```
1
2 void ntt(vector<Mod> &a, bool inv, Mod primitive_root) {
3     int n = a.size();
4     Mod root = primitive_root ^ (MOD - 1) / n;
5     vector<Mod> rt(n + 1, 1);
6     for (int i = 0; i < n; i++) rt[i + 1] = rt[i] * root;
7     fft_(n, a, rt, inv);
8 }
9 void fft(vector<complex<double>> &a, bool inv) {
10    int n = a.size();
11    vector<complex<double>> rt(n + 1);
12    double arg = acos(-1) * 2 / n;
13    for (int i = 0; i <= n; i++)
14        rt[i] = {cos(arg * i), sin(arg * i)};
15    fft_(n, a, rt, inv);
16 }
```

5.4. Fast Walsh-Hadamard Transform

Requires: Mod Struct

```
1
2 void fwht(vector<Mod> &a, bool inv) {
3     int n = a.size();
4     for (int d = 1; d < n; d <= 1)
5         for (int m = 0; m < n; m++)
6             if (!(m & d)) {
7                 inv ? a[m] -= a[m | d] : a[m] += a[m | d]; // AND
8                 inv ? a[m | d] -= a[m] : a[m | d] += a[m]; // OR
9                 Mod x = a[m], y = a[m | d];
10                a[m] = x + y, a[m | d] = x - y; // XOR
11            }
12     if (Mod iv = Mod(1) / n; inv) // XOR
13         for (Mod &i : a) i *= iv; // XOR
14 }
```

5.5. Subset Convolution

Requires: Mod Struct

```
1 #pragma GCC target("popcnt")
2 #include <immintrin.h>
3
4 void fwht(int n, vector<vector<Mod>> &a, bool inv) {
5     for (int h = 0; h < n; h++)
6         for (int i = 0; i < (1 << n); i++)
7             if (!(i & (1 << h)))
8                 for (int k = 0; k <= n; k++)
9                     inv ? a[i | (1 << h)][k] -= a[i][k]
10                        : a[i | (1 << h)][k] += a[i][k];
11 }
12 // c[k] = sum(popcnt(i & j) == sz && i | j == k) a[i] * b[j]
13 vector<Mod> subset_convolution(int n, int sz,
14                                 const vector<Mod> &a,
15                                 const vector<Mod> &b) {
16     int len = n + sz + 1, N = 1 << n;
17     vector<vector<Mod>> a(1 << n, vector<Mod>(len, 0)), b = a;
18     for (int i = 0; i < N; i++)
19         a[i][__mm_popcnt_u64(i)] = a[i];
20     b[i][__mm_popcnt_u64(i)] = b[i];
21     fwht(n, a, 0), fwht(n, b, 0);
22     for (int i = 0; i < N; i++) {
23         vector<Mod> tmp(len);
24         for (int j = 0; j < len; j++)
25             for (int k = 0; k <= j; k++)
26                 tmp[j] += a[i][k] * b[i][j - k];
27         a[i] = tmp;
28     }
29     fwht(n, a, 1);
30     vector<Mod> c(N);
31     for (int i = 0; i < N; i++)
32         c[i] = a[i][__mm_popcnt_u64(i) + sz];
33     return c;
34 }
```

5.6. Linear Recurrences

5.6.1. Berlekamp-Massey Algorithm

```
1 template <typename T>
2 vector<T> berlekamp_massey(const vector<T> &s) {
3     int n = s.size(), l = 0, m = 1;
4     vector<T> r(n), p(n);
5     r[0] = p[0] = 1;
6     T b = 1, d = 0;
7     for (int i = 0; i < n; i++) {
8         for (int j = 0; j <= l; j++) d += r[j] * s[i - j];
9         if ((d /= b) == 0) continue; // change if T is float
10        auto t = r;
11        for (int j = m; j < n; j++) r[j] -= d * p[j - m];
12        if (l * 2 <= i) l = i + 1 - l, b *= d, m = 0, p = t;
13    }
14    return r.resize(l + 1), reverse(r.begin(), r.end()), r;
15 }
```

5.6.2. Linear Recurrence Calculation

```
1 template <typename T> struct lin_rec {
2     using poly = vector<T>;
3     poly mul(poly a, poly b, poly m) {
4         int n = m.size();
5         poly r(n);
6         for (int i = n - 1; i >= 0; i--) {
7             r.insert(r.begin(), 0), r.pop_back();
8             T c = r[n - 1] + a[n - 1] * b[i];
9             // c /= m[n - 1]; if m is not monic
10            for (int j = 0; j < n; j++)
11                r[j] += a[j] * b[i] - c * m[j];
12        }
13        return r;
14 }
```

```

15 }
16 poly pow(poly p, ll k, poly m) {
17     poly r(m.size());
18     r[0] = 1;
19     for (; k; k >= 1, p = mul(p, p, m))
20         if (k & 1) r = mul(r, p, m);
21     return r;
22 }
23 T calc(poly t, poly r, ll k) {
24     int n = r.size();
25     poly p(n);
26     p[1] = 1;
27     poly q = pow(p, k, r);
28     T ans = 0;
29     for (int i = 0; i < n; i++) ans += t[i] * q[i];
30     return ans;
31 }

```

5.7. Matrices

5.7.1. Determinant

Requires: Mod Struct

```

1
2 Mod det(vector<vector<Mod>> a) {
3     int n = a.size();
4     Mod ans = 1;
5     for (int i = 0; i < n; i++) {
6         int b = i;
7         for (int j = i + 1; j < n; j++) {
8             if (a[j][i] != 0) {
9                 b = j;
10                break;
11            }
12            if (i != b) swap(a[i], a[b]), ans = -ans;
13            ans *= a[i][i];
14            if (ans == 0) return 0;
15            for (int j = i + 1; j < n; j++) {
16                Mod v = a[j][i] / a[i][i];
17                if (v != 0)
18                    for (int k = i + 1; k < n; k++)
19                        a[j][k] -= v * a[i][k];
20            }
21        }
22    }
23    return ans;
24 }

```

```

1 double det(vector<vector<double>> a) {
2     int n = a.size();
3     double ans = 1;
4     for (int i = 0; i < n; i++) {
5         int b = i;
6         for (int j = i + 1; j < n; j++)
7             if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
8         if (i != b) swap(a[i], a[b]), ans = -ans;
9         ans *= a[i][i];
10        if (ans == 0) return 0;
11        for (int j = i + 1; j < n; j++) {
12            double v = a[j][i] / a[i][i];
13            if (v != 0)
14                for (int k = i + 1; k < n; k++)
15                    a[j][k] -= v * a[i][k];
16        }
17    }
18    return ans;
19 }

```

5.7.2. Inverse

```

1
2 // Returns rank.
3 // Result is stored in A unless singular (rank < n).
4 // For prime powers, repeatedly set
5 // A^{-1} = A^{-1} * (2I - A^*A^{-1}) (mod p^k)
6 // where A^{-1} starts as the inverse of A mod p,
7 // and k is doubled in each step.
8
9 int matInv(vector<vector<double>> &A) {
10    int n = sz(A);
11    vi col(n);
12    vector<vector<double>> tmp(n, vector<double>(n));
13    rep(i, 0, n) tmp[i][i] = 1, col[i] = i;
14
15    rep(i, 0, n) {
16        int r = i, c = i;
17        rep(j, i, n)
18            rep(k, i, n) if (fabs(A[j][k]) > fabs(A[r][c])) r = j, c = k;
19        if (fabs(A[r][c]) < 1e-12) return i;
20    }
21 }

```

```

23     A[i].swap(A[r]);
24     tmp[i].swap(tmp[r]);
25     rep(j, 0, n) swap(A[j][i], A[j][c]);
26     swap(tmp[j][i], tmp[j][c]);
27     swap(col[i], col[c]);
28     double v = A[i][i];
29     rep(j, i + 1, n) {
30         double f = A[j][i] / v;
31         A[j][i] = 0;
32         rep(k, i + 1, n) A[j][k] -= f * A[i][k];
33         rep(k, 0, n) tmp[j][k] -= f * tmp[i][k];
34     }
35     rep(j, i + 1, n) A[i][j] /= v;
36     rep(j, 0, n) tmp[i][j] /= v;
37     A[i][i] = 1;
38 }
39
40 for (int i = n - 1; i > 0; --i) rep(j, 0, i) {
41     double v = A[j][i];
42     rep(k, 0, n) tmp[j][k] -= v * tmp[i][k];
43 }
44
45 rep(i, 0, n) rep(j, 0, n) A[col[i]][col[j]] = tmp[i][j];
46
47 }
48
49 int matInv_mod(vector<vector<ll>> &A) {
50     int n = sz(A);
51     vi col(n);
52     vector<vector<ll>> tmp(n, vector<ll>(n));
53     rep(i, 0, n) tmp[i][i] = 1, col[i] = i;
54
55     rep(i, 0, n) {
56         int r = i, c = i;
57         rep(j, i, n) rep(k, i, n) if (A[j][k]) {
58             r = j;
59             c = k;
60             goto found;
61         }
62         return i;
63     }
64     found:
65     A[i].swap(A[r]);
66     tmp[i].swap(tmp[r]);
67     rep(j, 0, n) swap(A[j][i], A[j][c]);
68     swap(tmp[j][i], tmp[j][c]);
69     swap(col[i], col[c]);
70     ll v = modpow(A[i][i], mod - 2);
71     rep(j, i + 1, n) {
72         ll f = A[j][i] * v % mod;
73         A[j][i] = 0;
74         rep(k, i + 1, n) A[j][k] =
75             (A[j][k] - f * A[i][k]) % mod;
76         rep(k, 0, n) tmp[j][k] =
77             (tmp[j][k] - f * tmp[i][k]) % mod;
78     }
79     rep(j, i + 1, n) A[i][j] = A[i][j] * v % mod;
80     rep(j, 0, n) tmp[i][j] = tmp[i][j] * v % mod;
81     A[i][i] = 1;
82 }
83
84 for (int i = n - 1; i > 0; --i) rep(j, 0, i) {
85     ll v = A[j][i];
86     rep(k, 0, n) tmp[j][k] =
87         (tmp[j][k] - v * tmp[i][k]) % mod;
88 }
89
90 rep(i, 0, n) rep(j, 0, n) A[col[i]][col[j]] =
91     tmp[i][j] % mod + (tmp[i][j] < 0 ? mod : 0);
92
93 }

```

5.7.3. Characteristic Polynomial

```

1
2
3 // calculate det(a - xI)
4 template <typename T>
5 vector<T> CharacteristicPolynomial(vector<vector<T>> a) {
6     int N = a.size();
7
8     for (int j = 0; j < N - 2; j++) {
9         for (int i = j + 1; i < N; i++) {
10             if (a[i][j] != 0) {
11                 swap(a[j + 1], a[i]);
12                 for (int k = 0; k < N; k++)
13                     swap(a[k][j + 1], a[k][i]);
14                 break;
15             }
16         }
17     }
18     if (a[j + 1][j] != 0) {
19         T inv = T(1) / a[j + 1][j];
20         for (int i = j + 2; i < N; i++) {
21             a[i] *= inv;
22         }
23     }
24 }

```

```

21     if (a[i][j] == 0) continue;
22     T coe = inv * a[i][j];
23     for (int l = j; l < N; l++)
24       a[i][l] -= coe * a[j + 1][l];
25     for (int k = 0; k < N; k++)
26       a[k][j + 1] += coe * a[k][i];
27   }
28 }

31 vector<vector<T>> p(N + 1);
32 p[0] = {T(1)};
33 for (int i = 1; i <= N; i++) {
34   p[i].resize(i + 1);
35   for (int j = 0; j < i; j++) {
36     p[i][j + 1] -= p[i - 1][j];
37     p[i][j] += p[i - 1][j] * a[i - 1][i - 1];
38   }
39   T x = 1;
40   for (int m = 1; m < i; m++) {
41     x *= -a[i - m][i - m - 1];
42     T coe = x * a[i - m - 1][i - 1];
43     for (int j = 0; j < i - m; j++)
44       p[i][j] += coe * p[i - m - 1][j];
45   }
46 }
47 return p[N];
}

```

5.7.4. Solve Linear Equation

```

1

3 typedef vector<double> vd;
4 const double eps = 1e-12;
5
6 // solves for x: A * x = b
7 int solvelinear(vector<vd> &A, vd &b, vd &x) {
8   int n = sz(A), m = sz(x), rank = 0, br, bc;
9   if (n) assert(sz(A[0]) == m);
10  vi col(m);
11  iota(all(col), 0);
12
13  rep(i, 0, n) {
14    double v, bv = 0;
15    rep(r, i, n) rep(c, i, m) if ((v = fabs(A[r][c])) > bv)
16      br = r,
17      bc = c, bv = v;
18    if (bv <= eps) {
19      rep(j, i, n) if (fabs(b[j]) > eps) return -1;
20      break;
21    }
22    swap(A[i], A[br]);
23    swap(b[i], b[br]);
24    swap(col[i], col[bc]);
25    rep(j, 0, n) swap(A[j][i], A[j][bc]);
26    bv = 1 / A[i][i];
27    rep(j, i + 1, n) {
28      double fac = A[j][i] * bv;
29      b[j] -= fac * b[i];
30      rep(k, i + 1, m) A[j][k] -= fac * A[i][k];
31    }
32    rank++;
33  }
34
35  x.assign(m, 0);
36  for (int i = rank; i--;) {
37    b[i] /= A[i][i];
38    x[col[i]] = b[i];
39    rep(j, 0, i) b[j] -= A[j][i] * b[i];
40  }
41  return rank; // (multiple solutions if rank < m)
}

```

5.8. Polynomial Interpolation

```

1

3 // returns a, such that a[0]x^0 + a[1]x^1 + a[2]x^2 + ...
4 // passes through the given points
5 typedef vector<double> vd;
6 vd interpolate(vd x, vd y, int n) {
7   vd res(n), temp(n);
8   rep(k, 0, n - 1) rep(i, k + 1, n) y[i] =
9     (y[i] - y[k]) / (x[i] - x[k]);
10  double last = 0;
11  temp[0] = 1;
12  rep(k, 0, n) rep(i, 0, n) {
13    res[i] += y[k] * temp[i];
14    swap(last, temp[i]);
15    temp[i] -= last * x[k];
16  }
17  return res;
}

```

5.9. Simplex Algorithm

```

1 // Two-phase simplex algorithm for solving linear programs
2 // of the form
3 //
4 // maximize      c^T x
5 // subject to    Ax <= b
6 //                  x >= 0
7 //
8 // INPUT: A -- an m x n matrix
9 //        b -- an m-dimensional vector
10 //       c -- an n-dimensional vector
11 //       x -- a vector where the optimal solution will be
12 //             stored
13 //
14 // OUTPUT: value of the optimal solution (infinity if
15 //         unbounded
16 //         above, nan if infeasible)
17 //
18 // To use this code, create an LPSolver object with A, b,
19 // and c as arguments. Then, call Solve(x).
20
21 typedef long double ld;
22 typedef vector<ld> vd;
23 typedef vector<vd> vvd;
24 typedef vector<int> vi;
25
26 const ld EPS = 1e-9;
27
28 struct LPSolver {
29   int m, n;
30   vi B, N;
31   vvd D;
32
33   LPSolver(const vvd &A, const vd &b, const vd &c)
34     : m(b.size()), n(c.size()), N(n + 1), B(m),
35       D(m + 2, vd(n + 2)) {
36     for (int i = 0; i < m; i++)
37       for (int j = 0; j < n; j++) D[i][j] = A[i][j];
38     for (int i = 0; i < m; i++) {
39       B[i] = n + i;
40       D[i][n] = -1;
41       D[i][n + 1] = b[i];
42     }
43     for (int j = 0; j < n; j++) {
44       N[j] = j;
45       D[m][j] = -c[j];
46     }
47     N[n] = -1;
48     D[m + 1][n] = 1;
49   }
50
51   void Pivot(int r, int s) {
52     double inv = 1.0 / D[r][s];
53     for (int i = 0; i < m + 2; i++) {
54       if (i != r)
55         for (int j = 0; j < n + 2; j++) {
56           if (j != s) D[i][j] -= D[r][j] * D[i][s] * inv;
57         }
58       if (j != s) D[r][j] *= inv;
59     }
60     for (int i = 0; i < m + 2; i++) {
61       if (i != r) D[i][s] *= -inv;
62       D[r][s] = inv;
63       swap(B[r], N[s]);
64     }
65
66   bool Simplex(int phase) {
67     int x = phase == 1 ? m + 1 : m;
68     while (true) {
69       int s = -1;
70       for (int j = 0; j <= n; j++) {
71         if (phase == 2 && N[j] == -1) continue;
72         if (s == -1 || D[x][j] < D[x][s] ||
73             D[x][j] == D[x][s] && N[j] < N[s])
74           s = j;
75       }
76       if (D[x][s] > -EPS) return true;
77       int r = -1;
78       for (int i = 0; i < m; i++) {
79         if (D[i][s] < EPS) continue;
80         if (r == -1 ||
81             D[i][n + 1] / D[i][s] < D[r][n + 1] / D[r][s] ||
82             (D[i][n + 1] / D[i][s]) ==
83             (D[r][n + 1] / D[r][s]) &&
84             B[i] < B[r])
85           r = i;
86       }
87       if (r == -1) return false;
88       Pivot(r, s);
89     }
90
91   ld Solve(vd &x) {
92     int r = 0;
93   }

```

```

93  for (int i = 1; i < m; i++) {
94    if (D[i][n + 1] < D[r][n + 1]) r = i;
95    if (D[r][n + 1] < -EPS) {
96      Pivot(r, n);
97      if (!Simplex(1) || D[m + 1][n + 1] < -EPS)
98        return numeric_limits<ld>::infinity();
99      for (int i = 0; i < m; i++)
100        if (B[i] == -1) {
101          int s = -1;
102          for (int j = 0; j <= n; j++)
103            if (s == -1 || D[i][j] < D[i][s] ||
104                D[i][j] == D[i][s] && N[j] < N[s])
105              s = j;
106          Pivot(i, s);
107        }
108      if (!Simplex(2)) return numeric_limits<ld>::infinity();
109      x = vd(n);
110      for (int i = 0; i < m; i++)
111        if (B[i] < n) x[B[i]] = D[i][n + 1];
112      return D[m][n + 1];
113    }
114  };
115
116 int main() {
117
118  const int m = 4;
119  const int n = 3;
120  ld _A[m][n] = {
121    {6, -1, 0}, {-1, -5, 0}, {1, 5, 1}, {-1, -5, -1}};
122  ld _b[m] = {10, -4, 5, -5};
123  ld _c[n] = {1, -1, 0};
124
125  vvd A(m);
126  vd b(_b, _b + m);
127  vd c(_c, _c + n);
128  for (int i = 0; i < m; i++) A[i] = vd(_A[i], _A[i] + n);
129
130  LPSolver solver(A, b, c);
131  vd x;
132  ld value = solver.Solve(x);
133
134  cerr << "VALUE: " << value << endl; // VALUE: 1.29032
135  cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
136  for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
137  cerr << endl;
138  return 0;
139}

```

6. Geometry

6.1. Point

```

1 template <typename T> struct P {
2   T x, y;
3   P(T x = 0, T y = 0) : x(x), y(y) {}
4   bool operator<(const P &p) const {
5     return tie(x, y) < tie(p.x, p.y);
6   }
7   bool operator==(const P &p) const {
8     return tie(x, y) == tie(p.x, p.y);
9   }
10  P operator-() const { return {-x, -y}; }
11  P operator+(P p) const { return {x + p.x, y + p.y}; }
12  P operator-(P p) const { return {x - p.x, y - p.y}; }
13  P operator*(T d) const { return {x * d, y * d}; }
14  P operator/(T d) const { return {x / d, y / d}; }
15  T dist2() const { return x * x + y * y; }
16  double len() const { return sqrt(dist2()); }
17  P unit() const { return *this / len(); }
18  friend T dot(P a, P b) { return a.x * b.x + a.y * b.y; }
19  friend T cross(P a, P b) { return a.x * b.y - a.y * b.x; }
20  friend T cross(P a, P b, P o) {
21    return cross(a - o, b - o);
22  }
23}; using pt = P<ll>;

```

6.1.1. Quaternion

```

1 constexpr double PI = 3.141592653589793;
2 constexpr double EPS = 1e-7;
3 struct Q {
4   using T = double;
5   T x, y, z, r;
6   Q(T r = 0) : x(0), y(0), z(0), r(r) {}
7   Q(T x, T y, T z, T r = 0) : x(x), y(y), z(z), r(r) {}
8   friend bool operator==(const Q &a, const Q &b) {
9     return (a - b).abs2() <= EPS;
10  }
11  friend bool operator!=(const Q &a, const Q &b) {
12    return !(a == b);
13  }

```

```

13  }
14  Q operator-() { return Q(-x, -y, -z, -r); }
15  Q operator+(const Q &b) const {
16    return Q(x + b.x, y + b.y, z + b.z, r + b.r);
17  }
18  Q operator-(const Q &b) const {
19    return Q(x - b.x, y - b.y, z - b.z, r - b.r);
20  }
21  Q operator*(const T &t) const {
22    return Q(x * t, y * t, z * t, r * t);
23  }
24  Q operator*(const Q &b) const {
25    return Q(r * b.x + x * b.r + y * b.z - z * b.y,
26             r * b.y - x * b.z + y * b.r + z * b.x,
27             r * b.z + x * b.y - y * b.x + z * b.r,
28             r * b.r - x * b.x - y * b.y - z * b.z);
29  }
30  Q operator/(const Q &b) const { return *this * b.inv(); }
31  T abs2() const { return r * r + x * x + y * y + z * z; }
32  T len() const { return sqrt(abs2()); }
33  Q conj() const { return Q(-x, -y, -z, r); }
34  Q unit() const { return *this * (1.0 / len()); }
35  Q inv() const { return conj() * (1.0 / abs2()); }
36  friend T dot(Q a, Q b) {
37    return a.x * b.x + a.y * b.y + a.z * b.z;
38  }
39  friend Q cross(Q a, Q b) {
40    return Q(a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z,
41             a.x * b.y - a.y * b.x);
42  }
43  friend Q rotation_around(Q axis, T angle) {
44    return axis.unit() * sin(angle / 2) + cos(angle / 2);
45  }
46  Q rotated_around(Q axis, T angle) {
47    Q u = rotation_around(axis, angle);
48    return u * *this / u;
49  }
50  friend Q rotation_between(Q a, Q b) {
51    a = a.unit(), b = b.unit();
52    if (a == -b) {
53      // degenerate case
54      Q ortho = abs(a.y) > EPS ? cross(a, Q(1, 0, 0))
55                                : cross(a, Q(0, 1, 0));
56      return rotation_around(ortho, PI);
57    }
58    return (a * (a + b)).conj();
59  }
}

```

6.1.2. Spherical Coordinates

```

1 struct car_p {
2   double x, y, z;
3 };
4 struct sph_p {
5   double r, theta, phi;
6 };
7 sph_p conv(car_p p) {
8   double r = sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
9   double theta = asin(p.y / r);
10  double phi = atan2(p.y, p.x);
11  return {r, theta, phi};
12 }
13 car_p conv(sph_p p) {
14   double x = p.r * cos(p.theta) * sin(p.phi);
15   double y = p.r * cos(p.theta) * cos(p.phi);
16   double z = p.r * sin(p.theta);
17   return {x, y, z};
18 }

```

6.2. Segments

```

1 // for non-collinear ABCD, if segments AB and CD intersect
2 bool intersects(pt a, pt b, pt c, pt d) {
3   if (cross(b, c, a) * cross(b, d, a) > 0) return false;
4   if (cross(d, a, c) * cross(d, b, c) > 0) return false;
5   return true;
6 }
7 // the intersection point of lines AB and CD
8 pt intersect(pt a, pt b, pt c, pt d) {
9   auto x = cross(b, c, a), y = cross(b, d, a);
10  if (x == y) {
11    // if(abs(x, y) < 1e-8) {
12    // is parallel
13  } else {
14    return d * (x / (x - y)) - c * (y / (x - y));
15  }
}

```

6.3. Convex Hull

```

1 // returns a convex hull in counterclockwise order

```

```
// for a non-strict one, change cross >= to >
3 vector<pt> convex_hull(vector<pt> p) {
4     sort(ALL(p));
5     if (p[0] == p.back()) return {p[0]};
6     int n = p.size(), t = 0;
7     vector<pt> h(n + 1);
8     for (int _ = 2, s = 0; _--; s = --t, reverse(ALL(p)))
9         for (pt i : p) {
10             while (t > s + 1 && cross(i, h[t - 1], h[t - 2]) >= 0)
11                 t--;
12             h[t++] = i;
13         }
14     return h.resize(t), h;
15 }
```

6.3.1. 3D Hull

```
1
3 typedef Point3D<double> P3;
5 struct PR {
6     void ins(int x) { (a == -1 ? a : b) = x; }
7     void rem(int x) { (a == x ? a : b) = -1; }
8     int cnt() { return (a != -1) + (b != -1); }
9     int a, b;
10 };
11
12 struct F {
13     P3 q;
14     int a, b, c;
15 };
16
17 vector<F> hull3d(const vector<P3> &A) {
18     assert(sz(A) >= 4);
19     vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
20 #define E(x, y) E[f.x][f.y]
21     vector<F> FS;
22     auto mf = [&](int i, int j, int k, int l) {
23         P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
24         if (q.dot(A[l]) > q.dot(A[i])) q = q * -1;
25         F f{q, i, j, k};
26         E(a, b).ins(k);
27         E(a, c).ins(j);
28         E(b, c).ins(i);
29         FS.push_back(f);
30     };
31     rep(i, 0, 4) rep(j, i + 1, 4) rep(k, j + 1, 4)
32         mf(i, j, k, 6 - i - j - k);
33
34     rep(i, 4, sz(A)) {
35         rep(j, 0, sz(FS)) {
36             F f = FS[j];
37             if (f.q.dot(A[i]) > f.q.dot(A[f.a])) {
38                 E(a, b).rem(f.c);
39                 E(a, c).rem(f.b);
40                 E(b, c).rem(f.a);
41                 swap(FS[j--], FS.back());
42                 FS.pop_back();
43             }
44             int nw = sz(FS);
45             rep(j, 0, nw) {
46                 F f = FS[j];
47 #define C(a, b, c)
48                 if (E(a, b).cnt() != 2) mf(f.a, f.b, i, f.c);
49                 C(a, b, c);
50                 C(a, c, b);
51                 C(b, c, a);
52             }
53         }
54         for (F &it : FS)
55             if ((A[it.b] - A[it.a])
56                 .cross(A[it.c] - A[it.a])
57                 .dot(it.q) <= 0)
58                 swap(it.c, it.b);
59     }
60     return FS;
61 }
```

6.4. Angular Sort

```
1 auto angle_cmp = [](<const pt &a, <const pt &b> {
2     auto btm = [](<const pt &a> {
3         return a.y < 0 || (a.y == 0 && a.x < 0);
4     };
5     return make_tuple(btm(a), a.y * b.x, abs2(a)) <
6         make_tuple(btm(b), a.x * b.y, abs2(b));
7 });
8 void angular_sort(vector<pt> &p) {
9     sort(p.begin(), p.end(), angle_cmp);
10 }
```

6.5. Convex Polygon Minkowski Sum

```
1 // O(n) convex polygon minkowski sum
2 // must be sorted and counterclockwise
3 vector<pt> minkowski_sum(vector<pt> p, vector<pt> q) {
4     auto diff = [](<vector<pt> &c>) {
5         auto rcmp = [](<pt a, pt b> {
6             return pt{a.y, a.x} < pt{b.y, b.x};
7         });
8         rotate(c.begin(), min_element(ALL(c), rcmp), c.end());
9         c.push_back(c[0]);
10        vector<pt> ret;
11        for (int i = 1; i < c.size(); i++)
12            ret.push_back(c[i] - c[i - 1]);
13        return ret;
14    };
15    auto dp = diff(p), dq = diff(q);
16    pt cur = p[0] + q[0];
17    vector<pt> d(dp.size() + dq.size()), ret = {cur};
18 // include angle_cmp from angular-sort.cpp
19 merge(ALL(dp), ALL(dq), d.begin(), angle_cmp);
20 // optional: make ret strictly convex (UB if degenerate)
21 int now = 0;
22 for (int i = 1; i < d.size(); i++) {
23     if (cross(d[i], d[now]) == 0) d[now] = d[now] + d[i];
24     else d[++now] = d[i];
25 }
26 d.resize(now + 1);
27 // end optional part
28 for (pt v : d) ret.push_back(cur = cur + v);
29 return ret.pop_back(), ret;
30 }
```

6.6. Point In Polygon

```
1 bool on_segment(pt a, pt b, pt p) {
2     return cross(a, b, p) == 0 && dot((p - a), (p - b)) <= 0;
3 }
4 // p can be any polygon, but this is O(n)
5 bool inside(const vector<pt> &p, pt a) {
6     int cnt = 0, n = p.size();
7     for (int i = 0; i < n; i++) {
8         pt l = p[i], r = p[(i + 1) % n];
9         // change to return 0; for strict version
10        if (on_segment(l, r, a)) return 1;
11        cnt += ((a.y < l.y) - (a.y < r.y)) * cross(l, r, a) > 0;
12    }
13    return cnt;
14 }
```

6.6.1. Convex Version

```
1 // no preprocessing version
2 // p must be a strict convex hull, counterclockwise
3 // if point is inside or on border
4 bool is_inside(const vector<pt> &c, pt p) {
5     int n = c.size(), l = 1, r = n - 1;
6     if (cross(c[0], c[1], p) < 0) return false;
7     if (cross(c[n - 1], c[0], p) < 0) return false;
8     while (l < r - 1) {
9         int m = (l + r) / 2;
10        T a = cross(c[0], c[m], p);
11        if (a > 0) l = m;
12        else if (a < 0) r = m;
13        else return dot(c[0] - p, c[m] - p) <= 0;
14    }
15    if (l == r) return dot(c[0] - p, c[l] - p) <= 0;
16    else return cross(c[l], c[r], p) >= 0;
17 }
18
19 // with preprocessing version
20 vector<pt> vecs;
21 pt center;
22 // p must be a strict convex hull, counterclockwise
23 // BEWARE OF OVERFLOWS!
24 void preprocess(vector<pt> p) {
25     for (auto &v : p) v = v * 3;
26     center = p[0] + p[1] + p[2];
27     center.x /= 3, center.y /= 3;
28     for (auto &v : p) v = v - center;
29     vecs = (angular_sort(p), p);
30 }
31 bool intersect_strict(pt a, pt b, pt c, pt d) {
32     if (cross(b, c, a) * cross(b, d, a) > 0) return false;
33     if (cross(d, a, c) * cross(d, b, c) >= 0) return false;
34     return true;
35 }
36 // if point is inside or on border
37 bool query(pt p) {
38     p = p * 3 - center;
39     auto pr = upper_bound(ALL(vecs), p, angle_cmp);
40     if (pr == vecs.end()) pr = vecs.begin();
41     auto pl = (pr == vecs.begin()) ? vecs.back() : *(pr - 1);
42     return !intersect_strict({0, 0}, p, pl, *pr);
43 }
```

6.7. Closest Pair

```

1 vector<pll> p; // sort by x first!
2 bool cmpy(const pll &a, const pll &b) const {
3     return a.y < b.y;
}
4 ll sq(ll x) { return x * x; }
5 // returns (minimum dist)^2 in [l, r)
6 ll solve(int l, int r) {
7     if (r - l <= 1) return 1e18;
8     int m = (l + r) / 2;
9     ll mid = p[m].x, d = min(solve(l, m), solve(m, r));
10    auto pb = p.begin();
11    inplace_merge(pb + l, pb + m, pb + r, cmpy);
12    vector<pll> s;
13    for (int i = l; i < r; i++)
14        if (sq(p[i].x - mid) < d) s.push_back(p[i]);
15    for (int i = 0; i < s.size(); i++)
16        for (int j = i + 1;
17             j < s.size() && sq(s[j].y - s[i].y) < d; j++)
18            d = min(d, dis(s[i], s[j]));
19    return d;
20}
21

```

6.8. Minimum Enclosing Circle

```

1
3 typedef Point<double> P;
4 double ccRadius(const P &A, const P &B, const P &C) {
5     return (B - A).dist() * (C - B).dist() * (A - C).dist() /
6         abs((B - A).cross(C - A)) / 2;
}
7 P ccCenter(const P &A, const P &B, const P &C) {
8     P b = C - A, c = B - A;
9     return A + (b * c.dist2() - c * b.dist2()).perp() /
10        b.cross(c) / 2;
}
11 pair<P, double> mec(vector<P> ps) {
12     shuffle(all(ps), mt19937(time(0)));
13     P o = ps[0];
14     double r = 0, EPS = 1 + 1e-8;
15     rep(i, 0, sz(ps)) if ((o - ps[i]).dist() > r * EPS) {
16         o = ps[i], r = 0;
17         rep(j, 0, i) if ((o - ps[j]).dist() > r * EPS) {
18             o = (ps[i] + ps[j]) / 2;
19             r = (o - ps[i]).dist();
20             rep(k, 0, j) if ((o - ps[k]).dist() > r * EPS) {
21                 o = ccCenter(ps[i], ps[j], ps[k]);
22                 r = (o - ps[i]).dist();
23             }
24         }
25     }
26     return {o, r};
27 }
28

```

6.9. Delaunay Triangulation

```

1
3 typedef Point<ll> P;
4 typedef struct Quad *Q;
5 typedef __int128_t lll; // (can be ll if coords are < 2e4)
6 P arb(LLONG_MAX, LLONG_MAX); // not equal to any other point
7
8 struct Quad {
9     bool mark;
10    Q o, rot;
11    P p;
12    P F() { return r()->p; }
13    Q r() { return rot->rot; }
14    Q prev() { return rot->o->rot; }
15    Q next() { return r()->prev(); }
16 };
17
18 bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
19     lll p2 = p.dist2(), A = a.dist2() - p2,
20     B = b.dist2() - p2, C = c.dist2() - p2;
21     return p.cross(a, b) * C + p.cross(b, c) * A +
22        p.cross(c, a) * B >
23        0;
}
24
25 Q makeEdge(P orig, P dest) {
26     Q q[] = {new Quad{0, 0, 0, orig}, new Quad{0, 0, 0, arb},
27               new Quad{0, 0, 0, dest}, new Quad{0, 0, 0, arb}};
28     rep(i, 0, 4) q[i]->o = q[-i & 3],
29                  q[i]->rot = q[(i + 1) & 3];
30     return *q;
}
31 void splice(Q a, Q b) {
32     swap(a->o->rot->o, b->o->rot->o);
33     swap(a->o, b->o);
}
34 Q connect(Q a, Q b) {
35

```

```

37     Q q = makeEdge(a->F(), b->p);
38     splice(q, a->next());
39     splice(q->r(), b);
40     return q;
}
41
42 pair<Q, Q> rec(const vector<P> &s) {
43     if (sz(s) <= 3) {
44         Q a = makeEdge(s[0], s[1]),
45             b = makeEdge(s[1], s.back());
46         if (sz(s) == 2) return {a, a->r()};
47         splice(a->r(), b);
48         auto side = s[0].cross(s[1], s[2]);
49         Q c = side ? connect(b, a) : 0;
50         return {side < 0 ? c->r() : a, side < 0 ? c : b->r()};
}
51
52 #define H(e) e->F(), e->p
53 #define valid(e) (e->F().cross(H(base)) > 0)
54 Q A, B, ra, rb;
55 int half = sz(s) / 2;
56 tie(ra, A) = rec({all(s) - half});
57 tie(B, rb) = rec({sz(s) - half + all(s)});
58 while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
59        (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
60 Q base = connect(B->r(), A);
61 if (A->p == ra->p) ra = base->r();
62 if (B->p == rb->p) rb = base;
63
64 #define DEL(e, init, dir)
65 Q e = init->dir;
66 if (valid(e))
67     while (circ(e->dir->F(), H(base), e->F())) {
68         Q t = e->dir;
69         splice(e, e->prev());
70         splice(e->r(), e->r()->prev());
71         e = t;
72     }
73 for (;;) {
74     DEL(LC, base->r(), o);
75     DEL(RC, base, prev());
76     if (!valid(LC) && !valid(RC)) break;
77     if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
78         base = connect(RC, base->r());
79     else base = connect(base->r(), LC->r());
80 }
81 return {ra, rb};
}
82
83 // returns [A_0, B_0, C_0, A_1, B_1, ...]
84 // where A_i, B_i, C_i are counter-clockwise triangles
85 vector<P> triangulate(vector<P> pts) {
86     sort(all(pts));
87     assert(unique(all(pts)) == pts.end());
88     if (sz(pts) < 2) return {};
89     Q e = rec(pts).first;
90     vector<Q> q = {e};
91     int qi = 0;
92     while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
93     #define ADD
94     {
95         Q c = e;
96         do {
97             c->mark = 1;
98             pts.push_back(c->p);
99             q.push_back(c->r());
100            c = c->next();
101        } while (c != e);
102    }
103    ADD;
104    pts.clear();
105    while (qi < sz(q))
106        if (!(e = q[qi++])->mark) ADD;
107    return pts;
}
108

```

6.9.1. Slower Version

```

1
3 template <class P, class F>
4 void delaunay(vector<P> &ps, F trifun) {
5     if (sz(ps) == 3) {
6         int d = (ps[0].cross(ps[1], ps[2]) < 0);
7         trifun(0, 1 + d, 2 - d);
}
8     vector<P3> p3;
9     for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
10    if (sz(ps) > 3)
11        for (auto t : hull3d(p3))
12            if ((p3[t.b] - p3[t.a])
13                .cross(p3[t.c] - p3[t.a])
14                .dot(P3(0, 0, 1)) < 0)
15                trifun(t.a, t.c, t.b);
}
16

```

17 }

6.10. Half Plane Intersection

```

1 struct Line {
2     Point P;
3     Vector v;
4     bool operator<(const Line &b) const {
5         return atan2(v.y, v.x) < atan2(b.v.y, b.v.x);
6     }
7 };  

8 bool OnLeft(const Line &L, const Point &p) {
9     return Cross(L.v, p - L.P) > 0;
10 }  

11 Point GetIntersection(Line a, Line b) {
12     Vector u = a.P - b.P;
13     Double t = Cross(b.v, u) / Cross(a.v, b.v);
14     return a.P + a.v * t;
15 }  

16 int HalfplaneIntersection(Line *L, int n, Point *poly) {
17     sort(L, L + n);  

18  

19     int first, last;
20     Point *p = new Point[n];
21     Line *q = new Line[n];
22     q[first = last = 0] = L[0];
23     for (int i = 1; i < n; i++) {
24         while (first < last && !OnLeft(L[i], p[last - 1]))
25             last--;
26         while (first < last && !OnLeft(L[i], p[first])) first++;
27         q[++last] = L[i];
28         if (fabs(Cross(q[last].v, q[last - 1].v)) < EPS) {
29             last--;
30             if (OnLeft(q[last], L[i].P)) q[last] = L[i];
31         }
32         if (first < last)
33             p[last - 1] = GetIntersection(q[last - 1], q[last]);
34     }
35     while (first < last && !OnLeft(q[first], p[last - 1]))
36         last--;
37     if (last - first <= 1) return 0;
38     p[last] = GetIntersection(q[last], q[first]);  

39  

40     int m = 0;
41     for (int i = first; i <= last; i++) poly[m++] = p[i];
42 }  

43 }
```

7. Strings

7.1. Knuth-Morris-Pratt Algorithm

```

1  

3 vector<int> pi(const string &s) {
4     vector<int> p(s.size());
5     for (int i = 1; i < s.size(); i++) {
6         int g = p[i - 1];
7         while (g && s[i] != s[g]) g = p[g - 1];
8         p[i] = g + (s[i] == s[g]);
9     }
10 }  

11 vector<int> match(const string &s, const string &pat) {
12     vector<int> p = pi(pat + '\0' + s), res;
13     for (int i = p.size() - s.size(); i < p.size(); i++)
14         if (p[i] == pat.size())
15             res.push_back(i - 2 * pat.size());
16     return res;
17 }
```

7.2. Aho-Corasick Automaton

```

1 struct Aho_Corasick {
2     static const int maxc = 26, maxn = 4e5;
3     struct NODES {
4         int Next[maxc], fail, ans;
5     };
6     NODES T[maxn];
7     int top, qtop, q[maxn];
8     int get_node(const int &fail) {
9         fill_n(T[top].Next, maxc, 0);
10        T[top].fail = fail;
11        T[top].ans = 0;
12        return top++;
13    }
14    int insert(const string &s) {
15        int ptr = 1;
16        for (char c : s) { // change char id
17            c -= 'a';
18            if (!T[ptr].Next[c]) T[ptr].Next[c] = get_node(ptr);
19            ptr = T[ptr].Next[c];
20        }
21    }
22    void search(const string &s) {
23        int ptr = 1;
24        for (char c : s) {
25            if (!T[ptr].Next[c]) break;
26            ptr = T[ptr].Next[c];
27        }
28        if (T[ptr].ans) cout << T[ptr].ans << endl;
29    }
30 };
31 
```

```

21     }
22     return ptr;
23 } // return ans_last_place
24 void build_fail(int ptr) {
25     int tmp;
26     for (int i = 0; i < maxc; i++) {
27         if (T[ptr].Next[i]) {
28             tmp = T[ptr].fail;
29             while (tmp != 1 && !T[tmp].Next[i])
30                 tmp = T[tmp].fail;
31             if (T[tmp].Next[i] != T[ptr].Next[i])
32                 T[T[ptr].Next[i]].fail = tmp;
33             q[qtop++] = T[ptr].Next[i];
34         }
35     }
36 } void AC_auto(const string &s) {
37     int ptr = 1;
38     for (char c : s) {
39         while (ptr != 1 && !T[ptr].Next[c]) ptr = T[ptr].fail;
40         if (T[ptr].Next[c]) {
41             ptr = T[ptr].Next[c];
42             T[ptr].ans++;
43         }
44     }
45 } void Solve(string &s) {
46     for (char &c : s) // change char id
47         c -= 'a';
48     for (int i = 0; i < qtop; i++) build_fail(q[i]);
49     AC_auto(s);
50     for (int i = qtop - 1; i > -1; i--)
51         T[T[q[i]].fail].ans += T[q[i]].ans;
52 } void reset() {
53     qtop = top = q[0] = 1;
54     get_node(1);
55 } AC;
56 // usage example
57 string s, S;
58 int n, t, ans_place[50000];
59 int main() {
60     Tie cin >> t;
61     while (t--) {
62         AC.reset();
63         cin >> S >> n;
64         for (int i = 0; i < n; i++) {
65             cin >> s;
66             ans_place[i] = AC.insert(s);
67         }
68     }
69     AC.Solve(S);
70     for (int i = 0; i < n; i++)
71         cout << AC.T[ans_place[i]].ans << '\n';
72 }
73 }
```

7.3. Suffix Array

```

1  

3 // sa[i]: starting index of suffix at rank i
4 //          0-indexed, sa[0] = n (empty string)
5 // lcp[i]: lcp of sa[i] and sa[i - 1], lcp[0] = 0
6 struct SuffixArray {
7     vector<int> sa, lcp;
8     SuffixArray(string &s,
9                 int lim = 256) { // or basic_string<int>
10        int n = sz(s) + 1, k = 0, a, b;
11        vector<int> x(all(s) + 1), y(n), ws(max(n, lim)),
12        rank(n);
13        sa = lcp = y, iota(all(sa), 0);
14        for (int j = 0, p = 0; p < n;
15             j = max(1, j * 2), lim = p) {
16            p = j, iota(all(y), n - j);
17            for (int i = 0; i < n; i++) {
18                if (sa[i] >= j) y[p++] = sa[i] - j;
19                fill(all(ws), 0);
20            }
21            for (int i = 0; i < n; i++) ws[x[i]]++;
22            for (int i = 1; i < lim; i++) ws[i] += ws[i - 1];
23            for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
24            swap(x, y), p = 1, x[sa[0]] = 0;
25            for (int i = 1; i < n; i++)
26                a = sa[i - 1], b = sa[i],
27                x[b] = (y[a] == y[b] && y[a + j] == y[b + j])
28                    ? p - 1 : p++;
29        }
30        for (int i = 1; i < n; i++) rank[sa[i]] = i;
31        for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
32            for (k && k--, j = sa[rank[i] - 1];
33                 s[i + k] == s[j + k]; k++);
34    }
35 }
```

```
37 }
```

7.4. Suffix Tree

```
1 struct SAM {
2     static const int maxc = 26; // char range
3     static const int maxn = 10010; // string len
4     struct Node {
5         Node *green, *edge[maxc];
6         int max_len, in, times;
7     } *root, *last, reg[maxn * 2];
8     int top;
9     Node *get_node(int _max) {
10        Node *re = &reg[top++];
11        re->in = 0, re->times = 1;
12        re->max_len = _max, re->green = 0;
13        for (int i = 0; i < maxc; i++) re->edge[i] = 0;
14        return re;
15    }
16    void insert(const char c) { // c in range [0, maxc)
17        Node *p = last;
18        last = get_node(p->max_len + 1);
19        while (p && !p->edge[c])
20            p->edge[c] = last, p = p->green;
21        if (!p) last->green = root;
22        else {
23            Node *pot_green = p->edge[c];
24            if ((pot_green->max_len) == (p->max_len + 1))
25                last->green = pot_green;
26            else {
27                Node *wish = get_node(p->max_len + 1);
28                wish->times = 0;
29                while (p && p->edge[c] == pot_green)
30                    p->edge[c] = wish, p = p->green;
31                for (int i = 0; i < maxc; i++)
32                    wish->edge[i] = pot_green->edge[i];
33                wish->green = pot_green->green;
34                pot_green->green = wish;
35                last->green = wish;
36            }
37        }
38    }
39    Node *q[maxn * 2];
40    int ql, qr;
41    void get_times(Node *p) {
42        ql = 0, qr = -1, reg[0].in = 1;
43        for (int i = 1; i < top; i++) reg[i].green->in++;
44        for (int i = 0; i < top; i++)
45            if (!reg[i].in) q[++qr] = &reg[i];
46        while (ql <= qr) {
47            q[ql]->green->times += q[ql]->in;
48            if (!(--q[ql]->green->in)) q[++qr] = q[ql]->green;
49            ql++;
50        }
51    }
52    void build(const string &s) {
53        top = 0;
54        root = last = get_node(0);
55        for (char c : s) insert(c - 'a'); // change char id
56        get_times(root);
57    }
58    // call build before solve
59    int solve(const string &s) {
60        Node *p = root;
61        for (char c : s)
62            if (!(p = p->edge[c - 'a'])) // change char id
63                return 0;
64        return p->times;
65    }
66 }
```

7.5. Cocke-Younger-Kasami Algorithm

```
1
2 struct rule {
3     // s -> xy
4     // if y == -1, then s -> x (unit rule)
5     int s, x, y, cost;
6 };
7 int state;
8 // state (id) for each letter (variable)
9 // lowercase letters are terminal symbols
10 map<char, int> rules;
11 vector<rule> cnf;
12 void init() {
13     state = 0;
14     rules.clear();
15     cnf.clear();
16 }
17 // convert a cfg rule to cnf (but with unit rules) and add
18 // it
19 //
```

```
21 void add_to_cnf(char s, const string &p, int cost) {
22     if (!rules.count(s)) rules[s] = state++;
23     for (char c : p)
24         if (!rules.count(c)) rules[c] = state++;
25     if (p.size() == 1) {
26         cnf.push_back({rules[s], rules[p[0]], -1, cost});
27     } else {
28         // length >= 3 -> split
29         int left = rules[s];
30         int sz = p.size();
31         for (int i = 0; i < sz - 2; i++) {
32             cnf.push_back({left, rules[p[i]], state, 0});
33             left = state++;
34         }
35         cnf.push_back(
36             {left, rules[p[sz - 2]], rules[p[sz - 1]], cost});
37     }
38
39 constexpr int MAXN = 55;
40 vector<long long> dp[MAXN][MAXN];
41 // unit rules with negative costs can cause negative cycles
42 vector<bool> neg_INF[MAXN][MAXN];
43
44 void relax(int l, int r, rule c, long long cost,
45           bool neg_c = 0) {
46     if (!neg_INF[l][r][c.s] &&
47         (neg_INF[l][r][c.x] || cost < dp[l][r][c.s])) {
48         if (neg_c || neg_INF[l][r][c.x]) {
49             dp[l][r][c.s] = 0;
50             neg_INF[l][r][c.s] = true;
51         } else {
52             dp[l][r][c.s] = cost;
53         }
54     }
55 }
56 void bellman(int l, int r, int n) {
57     for (int k = 1; k <= state; k++)
58         for (rule c : cnf)
59             if (c.y == -1)
60                 relax(l, r, c, dp[l][r][c.x] + c.cost, k == n);
61 }
62 void cyk(const string &s) {
63     vector<int> tok;
64     for (char c : s) tok.push_back(rules[c]);
65     for (int i = 0; i < tok.size(); i++) {
66         for (int j = 0; j < tok.size(); j++) {
67             dp[i][j] = vector<long long>(state + 1, INT_MAX);
68             neg_INF[i][j] = vector<bool>(state + 1, false);
69         }
70         dp[i][i][tok[i]] = 0;
71         bellman(i, i, tok.size());
72     }
73     for (int r = 1; r < tok.size(); r++) {
74         for (int l = r - 1; l >= 0; l--) {
75             for (int k = l; k < r; k++)
76                 for (rule c : cnf)
77                     if (c.y != -1)
78                         relax(l, r, c,
79                               dp[l][k][c.x] + dp[k + 1][r][c.y] +
80                               c.cost);
81             bellman(l, r, tok.size());
82         }
83     }
84 }
85
86 // usage example
87 int main() {
88     init();
89     add_to_cnf('S', "aSc", 1);
90     add_to_cnf('S', "BBB", 1);
91     add_to_cnf('S', "SB", 1);
92     add_to_cnf('B', "b", 1);
93     cyk("abbbbc");
94     // dp[0][s.size() - 1][rules[start]] = min cost to
95     // generate s
96     cout << dp[0][5][rules['S']] << '\n'; // 7
97     cyk("acbc");
98     cout << dp[0][3][rules['S']] << '\n'; // INT_MAX
99     add_to_cnf('S', "S", -1);
100    cyk("abbbbc");
101    cout << neg_INF[0][5][rules['S']] << '\n'; // 1
102 }
```

7.6. Z Value

```
1 int z[n];
2 void zval(string s) {
3     // z[i] => longest common prefix of s and s[i:], i > 0
4     int n = s.size();
5     z[0] = 0;
6     for (int b = 0, i = 1; i < n; i++) {
7         if (z[b] + b <= i) z[i] = 0;
8         else z[i] = min(z[i - b], z[b] + b - i);
9     }
10 }
```

```

9   while (s[i + z[i]] == s[z[i]]) z[i]++;
10  if (i + z[i] > b + z[b]) b = i;
11 }

```

```

47 } return SZ(St) - 2;
    };

```

7.7. Manacher's Algorithm

```

1 int z[n];
2 void manacher(string s) {
3     // z[i] => longest odd palindrome centered at i is
4     //      s[i - z[i] ... i + z[i]]
5     // to get all palindromes (including even length),
6     // insert a '#' between each s[i] and s[i + 1]
7     int n = s.size();
8     z[0] = 0;
9     for (int b = 0, i = 1; i < n; i++) {
10        if (z[b] + b >= i)
11            z[i] = min(z[2 * b - i], b + z[b] - i);
12        else z[i] = 0;
13        while (i + z[i] + 1 < n && i - z[i] - 1 >= 0 &&
14              s[i + z[i] + 1] == s[i - z[i] - 1])
15            z[i]++;
16        if (z[i] + i > z[b] + b) b = i;
17    }
}

```

7.8. Minimum Rotation

```

1 int min_rotation(string s) {
2     int a = 0, n = s.size();
3     s += s;
4     for (int b = 0; b < n; b++) {
5         for (int k = 0; k < n; k++) {
6             if (a + k == b || s[a + k] < s[b + k]) {
7                 b += max(0, k - 1);
8                 break;
9             }
10            if (s[a + k] > s[b + k]) {
11                a = b;
12                break;
13            }
14        }
15    }
16    return a;
17 }

```

7.9. Palindromic Tree

```

1
3 struct palindromic_tree {
4     struct node {
5         int next[26], fail, len;
6         int cnt,
7         num; // cnt: appear times, num: number of pal. suf.
8         node(int l = 0) : fail(0), len(l), cnt(0), num(0) {
9             for (int i = 0; i < 26; ++i) next[i] = 0;
10        }
11    };
12    vector<node> St;
13    vector<char> s;
14    int last, n;
15    palindromic_tree() : St(2), last(1), n(0) {
16        St[0].fail = 1, St[1].len = -1, s.pb(-1);
17    }
18    inline void clear() {
19        St.clear(), s.clear(), last = 1, n = 0;
20        St.pb(0), St.pb(-1);
21        St[0].fail = 1, s.pb(-1);
22    }
23    inline int get_fail(int x) {
24        while (s[n - St[x].len - 1] != s[n]) x = St[x].fail;
25        return x;
26    }
27    inline void add(int c) {
28        s.push_back(c == 'a'), ++n;
29        int cur = get_fail(last);
30        if (!St[cur].next[c]) {
31            int now = SZ(St);
32            St.pb(St[cur].len + 2);
33            St[now].fail = St[get_fail(St[cur].fail)].next[c];
34            St[cur].next[c] = now;
35            St[now].num = St[St[now].fail].num + 1;
36        }
37        last = St[cur].next[c], ++St[last].cnt;
38    }
39    inline void count() { // counting cnt
40        auto i = St.rbegin();
41        for (; i != St.rend(); ++i) {
42            St[i->fail].cnt += i->cnt;
43        }
44    }
45    inline int size() { // The number of diff. pal.
}

```

8. Debug List

- Pre-submit:
 - Did you make a typo when copying a template?
 - Test more cases if unsure.
 - Write a naive solution and check small cases.
 - Submit the correct file.
- General Debugging:
 - Read the whole problem again.
 - Have a teammate read the problem.
 - Have a teammate read your code.
 - Explain your solution to them (or a rubber duck).
 - Print the code and its output / debug output.
 - Go to the toilet.
- Wrong Answer:
 - Any possible overflows?
 - > `__int128` ?
 - Try `__ftrapv` or `#pragma GCC optimize("trapv")`
 - Floating point errors?
 - > `long double` ?
 - turn off math optimizations
 - check for `==`, `>=`, `acos(1.0000000001)` etc.
 - Did you forget to sort or unique?
 - Generate large and worst "corner" cases.
 - Check your `m` / `n`, `i` / `j` and `x` / `y`.
 - Are everything initialized or reset properly?
 - Are you sure about the STL thing you are using?
 - Read cppreference (should be available).
 - Print everything and run it on pen and paper.
- Time Limit Exceeded:
 - Calculate your time complexity again.
 - Does the program actually end?
 - Check for `while(q.size())` etc.
 - Test the largest cases locally.
 - Did you do unnecessary stuff?
 - e.g. pass vectors by value
 - e.g. `memset` for every test case
 - Is your constant factor reasonable?
- Runtime Error:
 - Check memory usage.
 - Forget to clear or destroy stuff?
 - > `vector::shrink_to_fit()`
 - Stack overflow?
 - Bad pointer / array access?
 - Try `__fsanitize=address`
 - Division by zero? NaN's?