

Math

matrizFibo

```

1 struct Matrix {
2     long long mat[2][2];
3     Matrix friend operator *(const Matrix &a, const
4         Matrix &b){
5         Matrix c;
6         for (int i = 0; i < 2; i++) {
7             for (int j = 0; j < 2; j++) {
8                 c.mat[i][j] = 0;
9                 for (int k = 0; k < 2; k++) {
10                     c.mat[i][j] += a.mat[i][k] * b.mat
11                         [k][j];
12                 }
13             }
14         }
15     };
16 Matrix matpow(Matrix base, long long n) {
17     Matrix ans{ {
18         {1, 0},
19         {0, 1}
20     } };
21     while (n) {
22         if(n&1)
23             ans = ans*base;
24         base = base*base;
25         n >>= 1;
26     }
27     return ans;
28 }
29 ll fib(int n) {
30     Matrix base{ {
31         {1, 1},
32         {1, 0}
33     } };
34     return matpow(base, n).mat[0][1];
35 }

```

Miscelanea

Plantilla

```

1 #include <bits/stdc++.h>
2 // #include <bits/extc++.h>
3 using namespace std;
4 // using namespace __gnu_pbds;
5 typedef long long ll;
6 typedef long double ld;
7
8 // typedef tree<pair<ll, ll>, null_type, less<pair<
9 //     ll, ll>>, rb_tree_tag,
10 //     tree_order_statistics_node_update>
11 //     ordered_set_men;
12 // typedef tree<int, null_type, greater<int>,
13 //     rb_tree_tag, tree_order_statistics_node_update>
14 //     ordered_set_may;
15
16 #define CRISTIANO RONALDO GANO 35 COPAS \
17     ios_base::sync_with_stdio(false); \
18     cin.tie(NULL); \
19     cout.tie(nullptr);
20
21 #define hola cout << "hola" << endl;
22 #define YES cout << "YES" << endl;
23 #define NO cout << "NO" << endl;
24 #define dbg(x) cout << #x << ": " << x << endl;
25 #define dbg2(x, y) cout << #x << ": " << x << " __ "
26 << #y << ": " << y << endl;
27
28 #define printvii(v_v) \
29     for (auto [x_x, y_y] : v_v) \
30     { \
31         cout << x_x << " " << y_y << endl; \
32     } \
33     cout << endl;
34
35 #define printst(st) \
36     for (auto num : st) \
37     cout << num << " "; \
38     cout << endl;
39
40 template <typename T>
41 void printv(T v)

```

```

33 {
34     for (auto x : v)
35         cout << x << " ";
36     cout << endl;
37 }
38
39 #define RAYA cout << "-----"
40     << endl;
41 // #define F first
42 // #define S second
43 const ll MOD = 1'000'000'007;
44 const vector<int> F = {0, 1, 0, -1};
45 const vector<int> C = {1, 0, -1, 0};
46
47 int main()
48 {
49     CRISTIANO RONALDO GANO 35 COPAS
50     //sol
51     return 0;
52 }

```

Datastructures

LowestCommonAncestor

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int n, m; //nodos, links
6 vector<int> h; //altura del nodo
7 vector<int> primera; //primera aparicion
8 // set<int> st;
9 vector<bool> vis;
10 vector<vector<int>> gf;
11 vector<int> nodo; //nodo
12
13 void dfs(int currt, int alt)
14 {
15     if (!vis[currt])
16     {
17         primera[currt] = h.size();
18     }
19 }

```

```

19 vis[currt] = 1;
20 h.push_back(alt);
21 nodo.push_back(currt);
22
23 for (int hijo : gf[currt])
24 {
25     if (!vis[hijo])
26     {
27         dfs(hijo, alt + 1);
28         h.push_back(alt);
29         nodo.push_back(currt);
30     }
31 }
32 }
33
34 int main()
35 {
36     vis.assign(n, 0);
37     primera.assign(n, -1);
38     gf.assign(n, vector<int>());
39     // st.clear();
40     h.clear();
41     nodo.clear();
42     //0i
43     dfs(0, 0);
44
45     //hacer un segmet tree sobre h que es la altura,
46     y responder cout<<nodo[res] donde res es el
47     indice del minimo en el rango, para minimo en
48     rango, donde l y r son los dos nodos
49
50     return 0;
51 }

```

QueueMin

```

1 struct quemin
2 {
3     stack<pair<int,int>> bo, to;
4     void push(int n)
5     {
6         if(bo.empty())
7             bo.push(mp(n, n));
8         else

```

```

9      bo.push(mp(n, min(bo.top().s, n)));
10  }
11  void pop()
12  {
13      if(to.empty())
14      {
15          while(!bo.empty())
16          {
17              if(to.empty())
18                  to.push(mp(bo.top().f, bo.top().f));
19              else
20                  to.push(mp(bo.top().f, min(bo.top().f, to.
21                      top().s)));
22              bo.pop();
23          }
24          to.pop();
25      }
26      int mini()
27      {
28          int mini = MOD;
29          if(!bo.empty())
30              mini = bo.top().s;
31          if(!to.empty())
32              mini = min(mini, to.top().s);
33          return mini;
34      }
35  };
36
37  struct quemin
38  {
39      pair<int,int> bo[100010], to[100010];
40      int boto = -1, toto = -1, ax;
41      void push(int n)
42      {
43          ax = boto + 1;
44          if(boto == -1)
45              bo[ax] = mp(n, n);
46          else
47              bo[ax] = mp(n, min(bo[boto].s, n));
48          boto++;
49      }
50      void pop()
51      {

```

```

52      if(toto == -1)
53      {
54          while(boto > -1)
55          {
56              ax = boto + 1;
57              if(toto == -1)
58                  to[ax] = mp(bo[boto].f, bo[boto].f);
59              else
60                  to[ax] = mp(bo[boto].f, min(bo[boto].f, to
61                      [toto].s));
62              toto++;
63              boto--;
64          }
65          if(toto > -1)
66              toto--;
67      }
68      int mini()
69      {
70          int mini = MOD;
71          if(boto > -1)
72              mini = bo[boto].s;
73          if(toto > -1)
74              mini = min(mini, to[toto].s);
75          return mini;
76      }
77  };

```

SegmentTree

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4  typedef long long ll;
5
6  struct Data
7  {
8      ll cant = 0;
9      Data() { cant = 1e18; }
10
11      Data(ll c) { cant = c; }
12  };
13

```

```

14 struct SegTree
15 {
16 private:
17     vector<Data> st;
18
19 public:
20     int sz;
21     Data merge(Data a, Data b)
22     {
23         return min(a.cant, b.cant);
24     }
25     void init(int n, vector<Data> v)
26     {
27         while (__builtin_popcount(n) != 1)
28         {
29             n++;
30         }
31         st.resize(2 * n, Data());
32         sz = n; // solo n, NO 2*n
33         for (int i = 0; i < (int)v.size(); i++)
34         {
35             st[n + i] = v[i];
36         }
37
38         for (int i = n - 1; i > 0; --i)
39         {
40             st[i] = merge(st[i << 1], st[(i << 1) +
41                             1]);
42         }
43
44     void updateTreeNode(int p, Data nuevoValor) // 0
45         i pos
46     {
47         st[p + sz] = nuevoValor;
48         p = p + sz;
49         for (int i = p; i > 1; i >>= 1)
50             st[i >> 1] = merge(st[i], st[i ^ 1]);
51     }
52
53     Data query(int nodo, int left_nodo, int
54               right_nodo, int l_q, int r_q) //
55         // query(1,0,st.sz-1,l_q,r_q) tipo->[l_q,r_q

```

```

56         ]0i
57     /*
58     los indices de los nodos empieza desde 1;
59     */
60     if (l_q <= left_nodo && right_nodo <= r_q)
61     {
62         return st[nodo];
63     }
64     if (l_q > right_nodo || left_nodo > r_q)
65     {
66         return Data();
67     }
68
69     int mitad = (left_nodo + right_nodo) / 2; //
70     //[l:r] --> [l:mitad] [mitad+1:r]
71     return merge(query(nodo * 2, left_nodo,
72                       mitad, l_q, r_q), query(nodo * 2 + 1,
73                       mitad + 1, right_nodo, l_q, r_q));
74 }
75 };

```

SparseTable

```

1 struct SparseTable {
2     int n;
3     vector<int> log2;
4     vector<vector<int>> st;
5
6     SparseTable(const vector<int>& a) {
7         n = a.size();
8         log2.resize(n + 1);
9         log2[1] = 0;
10        for (int i = 2; i <= n; i++)
11            log2[i] = log2[i/2] + 1;
12
13        int k = log2[n] + 1;
14        st.assign(n, vector<int>(k));
15
16        for (int i = 0; i < n; i++) st[i][0] = a[i];
17
18        for (int j = 1; j < k; j++)
19            for (int i = 0; i + (1 << j) <= n; i++)
20                st[i][j] = min(st[i][j-1], st[i + (1

```

```

21         << (j-1))][j-1]);
22     }
23     // Consulta de minimo en el rango [l, r]oi
24     int query(int l, int r) {
25         int j = log2[r - l + 1];
26         return min(st[l][j], st[r - (1 << j) + 1][j
27         ]);
28     };

```

UnionFind

```

1 struct unionFind {
2     vi p;
3     unionFind(int n) : p(n, -1) {}
4     int findParent(int v) {
5         if (p[v] == -1) return v;
6         return p[v] = findParent(p[v]);
7     }
8     bool join(int a, int b) {
9         a = findParent(a);
10        b = findParent(b);
11        if (a == b) return false;
12        p[a] = b;
13        return true;
14    }
15 };

```

fenwickTree

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 typedef long long ll;
5
6
7 struct Bit
8 {
9     private:
10        vector<ll> bit;
11        int sz;

```

```

12 public:
13
14     Bit(int n){bit.resize(n+1,0);this->sz=n;}
15
16     void init(vector<ll> vec)
17     {
18         for(int i=0;i<sz;i++)
19         {
20             update(i+1,vec[i]);
21         }
22     }
23
24     void update(int p,ll val)//1i
25     {
26         while(p<=sz)
27         {
28             bit[p]+=val;
29             p+=(p&(-p));
30         }
31     }
32     ll query(int p)//[1,p]1i
33     {
34         ll ans=0;
35         while(p>0)
36         {
37             ans+=bit[p];
38             p-=(p&-p);
39         }
40         return ans;
41     }
42
43     ll rquery(ll left,ll right)//[1,r]1i
44     {
45         return query(right)-query(left-1);
46     }
47 };

```

Strings

hashing

```

1 struct StrHash
2 { // Hash polinomial con exponentes decrecientes.

```

```

3   static constexpr ll ms[] = {1'000'000'007, 1'000
   '000'403};
4   static constexpr ll b = 500'000'000;
5   vector<ll> hs[2], bs[2];
6   StrHash(string const &s)
7   {
8       int n = s.length();
9       for(int k=0;k<2;k++)
10      {
11          hs[k].resize(n + 1), bs[k].resize(n + 1,
12          1);
13          for(int i=0;i<n;i++)
14          {
15              hs[k][i + 1] = (hs[k][i] * b + s[i])
16              % ms[k];
17              bs[k][i + 1] = bs[k][i] * b % ms[k];
18          }
19      }
20      ll get(int idx, int len) const
21      { // Hashes en 's[idx, idx+len)'
22          ll h[2];
23          for(int k=0;k<2;k++)
24          {
25              h[k] = hs[k][idx + len] - hs[k][idx] *
26              bs[k][len] % ms[k];
27              if (h[k] < 0)
28                  h[k] += ms[k];
29          }
30          return (h[0] << 32) | h[1];
31      }
32  };
33  //concat substrings(or strings) from non necesary
34  //two differents strings [idx,idx+lex)
35  ll concat_cross_hashes(const StrHash& A, int i1, int
36  len1, const StrHash& B, int i2, int len2)
37  {
38      ll res[2];
39      for (int k = 0; k < 2; ++k)
40      {
41          // hash de substring A[i1..i1+len1-1]
42          ll h1 = A.hs[k][i1 + len1] - A.hs[k][i1] * A
43          .bs[k][len1] % A.ms[k];

```

```

44          if (h1 < 0) h1 += A.ms[k];
45
46          // hash de substring B[i2..i2+len2-1]
47          ll h2 = B.hs[k][i2 + len2] - B.hs[k][i2] * B
48          .bs[k][len2] % B.ms[k];
49          if (h2 < 0) h2 += B.ms[k];
50
51          // combinacion: h1 * b^len2 + h2
52          res[k] = (h1 * A.bs[k][len2] + h2) % A.ms[k
53          ];
54      }
55      return (res[0] << 32) | res[1];
56  }
57  //0 indexed
58  ll h=StrHash("Hola").get(0,0+"Hola".size());

```

hashing2d

```

1  struct Hashing
2  {
3      vector<vector<int>> hs;
4      vector<int> PWX, PWY;
5      int n, m;
6      static const int PX = 3731, PY = 2999, mod =
7      998244353;
8      Hashing() {}
9      Hashing(vector<string> &s)
10     {
11         n = (int)s.size(), m = (int)s[0].size();
12         hs.assign(n + 1, vector<int>(m + 1, 0));
13         PWX.assign(n + 1, 1);
14         PWY.assign(m + 1, 1);
15         for (int i = 0; i < n; i++)
16             PWX[i + 1] = 1LL * PWX[i] * PX % mod;
17         for (int i = 0; i < m; i++)
18             PWY[i + 1] = 1LL * PWY[i] * PY % mod;
19         for (int i = 0; i < n; i++)
20         {
21             for (int j = 0; j < m; j++)
22                 hs[i + 1][j + 1] = s[i][j] - 'a' +

```

```

23         1;
24     }
25     for (int i = 0; i <= n; i++)
26     {
27         for (int j = 0; j < m; j++)
28         {
29             hs[i][j + 1] = (hs[i][j + 1] + 1LL *
30                 hs[i][j] * PY % mod) % mod;
31         }
32     }
33     for (int i = 0; i < n; i++)
34     {
35         for (int j = 0; j <= m; j++)
36         {
37             hs[i + 1][j] = (hs[i + 1][j] + 1LL *
38                 hs[i][j] * PX % mod) % mod;
39         }
40     }
41     int get_hash(int x1, int y1, int x2, int y2)
42     { // 1-indexed
43         assert(1 <= x1 && x1 <= x2 && x2 <= n);
44         assert(1 <= y1 && y1 <= y2 && y2 <= m);
45         x1--;
46         y1--;
47         int dx = x2 - x1, dy = y2 - y1;
48         return (1LL * (hs[x2][y2] - 1LL * hs[x2][y1]
49             * PWY[dy] % mod + mod) % mod -
50             1LL * (hs[x1][y2] - 1LL * hs[x1][y1]
51                 * PWY[dy] % mod + mod) % mod *
52                 PWX[dx] % mod + mod) %
53             mod;
54     }
55     int get_hash()
56     {
57         return get_hash(1, 1, n, m);
58     }
59 }

```

Geometry

Point

```

1  /*typedef double T;
2  typedef complex<T> pt;
3  #define x real()
4  #define y imag()*/
5
6  //typedef long long ll;
7  //typedef long double ll;
8
9  struct point
10 {
11     ll x, y;
12     point() {}
13     point(ll x, ll y): x(x), y(y) {}
14     point operator -(point p) {return point(x - p.x, y
15         - p.y);}
16     point operator +(point p) {return point(x + p.x, y
17         + p.y);}
18     ll sq() {return x * x + y * y;}
19     double abs() {return sqrt(sq());}
20     ll operator ^(point p) {return x * p.y - y * p.x;}
21     ll operator *(point p) {return x * p.x + y * p.y
22         ;}
23     point operator *(ll a) {return point(x * a, y *
24         a);}
25     bool operator <(const point& p) const {return x ==
26         p.x ? y < p.y : x < p.x;}
27     bool left(point a, point b) {return ((b - a) ^ (*
28         this - a)) >= 0;}
29     ostream& operator<<(ostream& os) {
30         return os << "(" << x << "," << y << ")";
31     }
32 }
33
34 };
35
36 void polarSort(vector<point>& v) {
37     sort(v.begin(), v.end(), [] (point a, point b) {
38         const point origin{0, 0};
39         bool ba = a < origin, bb = b < origin;
40         if (ba != bb) { return ba < bb; }
41         return (a^b) > 0;
42     });
43 }

```

```

35     });
36 }

```

convexHull

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  typedef long long ll;
5
6  struct pt
7  {
8      double x, y;
9      bool operator==(pt const &t) const
10     {
11         return x == t.x && y == t.y;
12     }
13 };
14
15 int orientation(pt a, pt b, pt c)
16 {
17     double v = a.x * (b.y - c.y) + b.x * (c.y - a.y) +
18             c.x * (a.y - b.y);
19     if (v < 0)
20         return -1; // clockwise
21     if (v > 0)
22         return +1; // counter-clockwise
23     return 0;
24 }
25
26 bool cw(pt a, pt b, pt c, bool include_collinear)
27 {
28     int o = orientation(a, b, c);
29     return o < 0 || (include_collinear && o == 0);
30 }
31
32 bool collinear(pt a, pt b, pt c) { return
33     orientation(a, b, c) == 0; }
34
35 void convex_hull(vector<pt> &a, bool
36     include_collinear = false)
37 {
38     pt p0 = *min_element(a.begin(), a.end(), [](pt a,
39     pt b)

```

```

35     { return make_pair(a.y, a.x) <
36         make_pair(b.y, b.x); });
37 sort(a.begin(), a.end(), [&p0](const pt &a, const
38     pt &b)
39     {
40         int o = orientation(p0, a, b);
41         if (o == 0)
42             return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)
43                 *(p0.y-a.y)
44                 < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)
45                     *(p0.y-b.y);
46         return o < 0; });
47 if (include_collinear)
48 {
49     int i = (int)a.size() - 1;
50     while (i >= 0 && collinear(p0, a[i], a.back()))
51         i--;
52     reverse(a.begin() + i + 1, a.end());
53 }
54
55 vector<pt> st;
56 for (int i = 0; i < (int)a.size(); i++)
57 {
58     while (st.size() > 1 && !cw(st[st.size() - 2],
59         st.back(), a[i], include_collinear))
60         st.pop_back();
61     st.push_back(a[i]);
62 }
63
64 if (include_collinear == false && st.size() == 2
65     && st[0] == st[1])
66     st.pop_back();
67
68 a = st;
69 }
70
71 vector<pt> a;
72
73 int main()
74 {
75     int n; cin >> n;
76     a.resize(n);
77     // leer como puntos

```



```

73 |     convex_hull(a,true); //true incluye colinear, false
    |         no lo hace
74 |
75 | }

```

Primalidad

RabinMiller

```

1 | #include <bits/stdc++.h>
2 |
3 | using namespace std;
4 |
5 |
6 | using u64 = uint64_t;
7 | using u128 = __uint128_t;
8 |
9 | u64 binpower(u64 base, u64 e, u64 mod) {
10 |     u64 result = 1;
11 |     base %= mod;
12 |     while (e) {
13 |         if (e & 1)
14 |             result = (u128)result * base % mod;
15 |         base = (u128)base * base % mod;
16 |         e >>= 1;
17 |     }
18 |     return result;
19 | }
20 |
21 | bool check_composite(u64 n, u64 a, u64 d, int s) {
22 |     u64 x = binpower(a, d, n);
23 |     if (x == 1 || x == n - 1)
24 |         return false;
25 |     for (int r = 1; r < s; r++) {
26 |         x = (u128)x * x % n;
27 |         if (x == n - 1)
28 |             return false;
29 |     }
30 |     return true;
31 | };
32 |
33 | bool MillerRabin(u64 n, int iter=5) { // returns
    |     true if n is probably prime, else returns false.

```

```

34 |     cout<<iter<<endl;
35 |     if (n < 4)
36 |         return n == 2 || n == 3;
37 |
38 |     int s = 0;
39 |     u64 d = n - 1;
40 |     while ((d & 1) == 0) {
41 |         d >>= 1;
42 |         s++;
43 |     }
44 |
45 |     for (int i = 0; i < iter; i++) {
46 |         int a = 2 + rand() % (n - 3);
47 |         if (check_composite(n, a, d, s))
48 |             return false;
49 |     }
50 |     return true;
51 | }
52 |
53 |
54 | int main()
55 | {
56 |
57 |     cout<<MillerRabin(100000001,30)<<endl;
58 |     return 0;
59 | }

```

pollardRho

```

1 | #include <iostream>
2 | #include <cstdlib>
3 | #include <cstdio>
4 | #include <cmath>
5 | #include <cassert>
6 | #include <map>
7 |
8 | using namespace std;
9 |
10 | typedef long long ll;
11 |
12 | #define forn(i, n) for (int i = 0; i < (int)(n); i++)
13 | #define forsn(i, s, n) for (int i = int(s); i < (int)

```

```

    )(n); i++)
14
15 // rabin miller
16
17 ll potlog(ll a, ll b, const ll M)
18 {
19     ll res = 1;
20     while (b)
21     {
22         if (b % 2)
23             res = (__int128(res) * a) % M;
24         a = (__int128(a) * a) % M;
25         b /= 2;
26     }
27     return res;
28 }
29
30 bool primo(ll n)
31 {
32     if (n < 2)
33         return false;
34     if (n == 2)
35         return true;
36     ll D = n - 1, S = 0;
37     while (D % 2 == 0)
38     {
39         D /= 2;
40         S++;
41     }
42     // n-1 = 2^S * D
43     static const int STEPS = 16;
44     forn(pasos, STEPS)
45     {
46         const ll A = 1 + rand() % (n - 1);
47         ll M = potlog(A, D, n);
48         if (M == 1 || M == (n - 1))
49             goto next;
50         forn(k, S - 1)
51         {
52             M = (__int128(M) * M) % n;
53             if (M == (n - 1))
54                 goto next;
55         }
56         return false;

```

```

57     next;;
58     }
59     return true;
60 }
61
62 // pollard's rho
63
64 ll mcd(ll a, ll b) { return (a == 0) ? b : mcd(b % a, a); }
65
66 ll factor(ll n)
67 {
68     static ll A, B;
69     A = 1 + rand() % (n - 1);
70     B = 1 + rand() % (n - 1);
71     #define f(x) ((__int128(x) * (x + B)) % n + A)
72     ll x = 2, y = 2, d = 1;
73     while (d == 1 || d == -1)
74     {
75         x = f(x);
76         y = f(f(y));
77         d = mcd(x - y, n);
78     }
79     return abs(d);
80 }
81
82 map<ll, ll> fact;
83
84 void factorize(ll n)
85 {
86     assert(n > 0);
87     while (n > 1 && !primo(n))
88     {
89         ll f;
90         do
91         {
92             f = factor(n);
93         } while (f == n);
94         n /= f;
95         factorize(f);
96         for (auto &it : fact)
97             while (n % it.first == 0)
98             {
99                 n /= it.first;

```

```

100         it.second++;
101     }
102 }
103 if (n > 1)
104     fact[n]++;
105 }
106
107 int main()
108 {
109     ll N;
110     while (cin >> N && N)
111     {
112         fact.clear();
113         factorize(N);
114         for (const auto &it : fact)
115             cout << it.first << "^" << it.second <<
116                 " ";
117         cout << endl;
118     }
119     return 0;

```

Graphs

isDag

```

1 vector<vector<int>> gf; // lista de adyacencia
2 vector<int> visited; // 0 = no visitado, 1 =
3     visitando, 2 = visitado
4 bool dfs(int u) {
5     visited[u] = 1; // visitando
6     for (int v : gf[u]) {
7         if (visited[v] == 1) return true; // ciclo
8             detectado
9         if (visited[v] == 0 && dfs(v)) return true;
10    }
11    visited[u] = 2; // visitado
12    return false;
13 }
14 bool isDAG(int n) {
15     visited.assign(n, 0);

```

```

16     for (int i = 0; i < n; i++)
17         if (visited[i] == 0 && dfs(i))
18             return false; // hay ciclo
19     return true; // no hay ciclos
20 }

```

toposort

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int n; // number of vertices
4 vector<vector<int>> gf;
5 vector<bool> vis;
6 vector<int> ans;
7
8 void dfs(int v) { // 0i
9     vis[v] = true;
10    for (int u : gf[v]) {
11        if (!vis[u]) {
12            dfs(u);
13        }
14    }
15    ans.push_back(v);
16 }
17
18 void topological_sort() {
19     vis.assign(n, false);
20     ans.clear();
21     for (int i = 0; i < n; ++i) {
22         if (!vis[i]) {
23             dfs(i);
24         }
25     }
26     reverse(ans.begin(), ans.end());
27 }

```

Overloads

pq

```

1 struct cmp
2 {

```

```

3      //mayor tiene prioridad
4      bool operator()(const int& a, const int& b)
5          const {
6              return a<b;
7          }
8      };
9      //priority_queue<Data,vector<Data>,cmp> pq

```

set

```

1 struct cmpST
2 {
3     //de menor a mayor
4     bool operator()(const int &a,const int &b)
5     {
6         return a<b;
7     }
8 };
9
10 //set<int,cmpST> st;

```