Contents

1 Basic

1.1 .vimrc

```
syntax on
set nu ai bs=2 sw=2 et ve=all cb=unnamed mouse=a ruler
    incsearch
```

2 Combinatorics

2.1 FFT

```
typedef complex<double> cpx;
const double PI = acos(-1);
vector<cpx> FFT(vector<cpx> &P, bool inv = 0) {
  assert(__builtin_popcount(P.size()) == 1);
  int lg = 31 - __builtin_clz(P.size()), n = 1 << lg;
      // == P.size();
  for (int j = 1, i = 0; j < n - 1; ++j) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j < i) swap(P[i], P[j]);
  } //bit reverse
  auto w1 = exp((2 - 4 * inv) * PI / n * cpx(0, 1)); //
       order is 1<<lg
  for (int i = 1; i <= lg; ++i) {
    auto wn = pow(w1, 1<<(lg - i)); // order is 1<<i
    for (int k = 0; k < (1<<lg); k += 1 << i) {
      cpx base = 1;
      for (int j = 0; j < (1 << i - 1); ++j, base =
          base * wn) {
        auto t = base * P[k + j + (1 << i - 1)];
        auto u = P[k + j];
        P[k + j] = u + t;
        P[k + j + (1 << i - 1)] = u - t;
      }
    }
  }
  if(inv)
    for (int i = 0; i < n; ++i) P[i] /= n;
  return P;
} //faster performance with calling by reference
```

2.2 FWT

```
vector<int> fast_OR_transform(vector<int> f, bool
    inverse) {
  for (int i = 0; (2 << i) <= f.size(); ++i)
    for (int j = 0; j < f.size(); j += 2 << i)
      for (int k = 0; k < (1 << i); ++k)
        f[j + k + (1 << i)] += f[j + k] * (inverse? -1
            : 1);
  return f;
}
vector<int> rev(vector<int> A) {
  for (int i = 0; i < A.size(); i += 2) swap(A[i], A[i
      ^ (A.size() - 1)]);
  return A;
}
vector<int> fast_AND_transform(vector<int> f, bool
    inverse) {
  return rev(fast_OR_transform(rev(f), inverse));
}
vector<int> fast_XOR_transform(vector<int> f, bool
    inverse) {
  for (int i = 0; (2 << i) <= f.size(); ++i)
    for (int j = 0; j < f.size(); j += 2 << i)
      for (int k = 0; k < (1 << i); ++k) {
        int u = f[j + k], v = f[j + k + (1 << i)];
        f[j + k + (1 << i)] = u - v, f[j + k] = u + v;
      }
  if (inverse) for (auto &a : f) a /= f.size();
  return f;
}
```

2.3 NTT

```
/* p == (a << n) + 1
   n    1 << n      p          a    root
   5    32          97         3    5
   6    64          193        3    5
   7    128         257        2    3
   8    256         257        1    3
   9    512         7681       15   17
   10   1024        12289      12   11
   11   2048        12289      6    11
   12   4096        12289      3    11
   13   8192        40961      5    3
   14   16384       65537      4    3
   15   32768       65537      2    3
   16   65536       65537      1    3
   17   131072      786433     6    10
   18   262144      786433     3    10 (605028353,
       2308, 3)
   19   524288      5767169    11   3
   20   1048576     7340033    7    3
   21   2097152     23068673   11   3
   22   4194304     104857601  25   3
   23   8388608     167772161  20   3
   24   16777216    167772161  10   3
   25   33554432    167772161  5    3 (1107296257, 33,
       10)
   26   67108864    469762049  7    3
   27   134217728   2013265921 15   31 */
LL root = 10, p = 786433, a = 3;
LL powM(LL x, LL b) {
  LL s = 1, m = x % p;
  for (; b; m = m * m % p, b >>= 1)
    if (b&1) s = s * m % p;
  return s;
}
vector<LL> NTT(vector<LL> P, bool inv = 0) {
  assert(__builtin_popcount(P.size()) == 1);
  int lg = 31 - __builtin_clz(P.size()), n = 1 << lg;
      // == P.size();
  for (int j = 1, i = 0; j < n - 1; ++j) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j < i) swap(P[i], P[j]);
  } //bit reverse
  LL w1 = powM(root, a * (inv? p - 2: 1)); // order is
      1<<lg
  for (LL i = 1; i <= lg; ++i) {
    LL wn = powM(w1, 1<<(lg - i)); // order is 1<<i
    for (int k = 0; k < (1<<lg); k += 1 << i) {
      LL base = 1;
      for (int j = 0; j < (1 << i - 1); ++j, base =
          base * wn % p) {
        LL t = base * P[k + j + (1 << i - 1)] % p;
        LL u = P[k + j] % p;
        P[k + j] = (u + t) % p;
        P[k + j + (1 << i - 1)] = (u - t + p) % p;
      }
    }
  }
  if(inv){
    LL invN = powM(n, p - 2);
    transform(P.begin(), P.end(), P.begin(), [&](LL a)
        {return a * invN % p;});
  }
  return P;
} //faster performance with calling by reference
```

2.4 permanent

```
typedef vector<vector<LL> > mat;
LL permanent(mat A) {
  LL n = A.size(), ans = 0, *tmp = new LL[n], add;
  for (int pgray = 0, s = 1, gray, i; s < 1 << n; ++s)
      {
    gray = s ^ s >> 1, add = 1;
    i = __builtin_ctz(pgray ^ gray);
    for (int j = 0; j < n; ++j)
      add *= tmp[j] += A[i][j] * (gray>>i&1 ? 1 : -1);
    ans += add * (s&1^n&1? -1 : 1), pgray = gray;
  }
  return ans;
}
// how many ways to put rooks on a matrix with 0,1 as
    constrain
// 1 - ok to put
// 0 - not ok to put
```

3 Data Structure

3.1 Heavy Light Decomposition

```
struct HLD {
  using Tree = vector<vector<int>>;
  vector<int> par, head, vid, len, inv;

  HLD(const Tree &g) : par(g.size()), head(g.size()),
      vid(g.size()), len(g.size()), inv(g.size()) {
    int k = 0;
    vector<int> size(g.size(), 1);
    function<void(int, int)> dfs_size = [&](int u, int
        p) {
      for (int v : g[u]) {
        if (v != p) {
          dfs_size(v, u);
          size[u] += size[v];
        }
      }
    };
    function<void(int, int, int)> dfs_dcmp = [&](int u,
         int p, int h) {
      par[u] = p;
      head[u] = h;
      vid[u] = k++;
      inv[vid[u]] = u;
      for (int v : g[u]) {
        if (v != p && size[u] < size[v] * 2) {
          dfs_dcmp(v, u, h);
        }
      }
      for (int v : g[u]) {
        if (v != p && size[u] >= size[v] * 2) {
          dfs_dcmp(v, u, v);
        }
      }
    };
    dfs_size(0, -1);
    dfs_dcmp(0, -1, 0);
    for (int i = 0; i < g.size(); ++i) {
      ++len[head[i]];
    }
  }

  template<typename T>
  void foreach(int u, int v, T f) {
    while (true) {
      if (vid[u] > vid[v]) {
        if (head[u] == head[v]) {
          f(vid[v] + 1, vid[u], 0);
          break;
        } else {
          f(vid[head[u]], vid[u], 1);
          u = par[head[u]];
        }
      } else {
        if (head[u] == head[v]) {
          f(vid[u] + 1, vid[v], 0);
          break;
        } else {
          f(vid[head[v]], vid[v], 0);
          v = par[head[v]];
        }
      }
    }
  }
};
```

4 Flow

4.1 CostFlow

```
template <class TF, class TC>
struct CostFlow {
  static const int MAXV = 205;
  static const TC INF = 0x3f3f3f3f;
  struct Edge {
    int v, r;
    TF f;
    TC c;
    Edge(int _v, int _r, TF _f, TC _c) : v(_v), r(_r),
        f(_f), c(_c) {}
  };
  int n, s, t, pre[MAXV], pre_E[MAXV], inq[MAXV];
  TF fl;
  TC dis[MAXV], cost;
  vector<Edge> E[MAXV];
  CostFlow(int _n, int _s, int _t) : n(_n), s(_s), t(_t
      ), fl(0), cost(0) {}
  void add_edge(int u, int v, TF f, TC c) {
    E[u].emplace_back(v, E[v].size(), f, c);
    E[v].emplace_back(u, E[u].size() - 1, 0, -c);
  }
  pair<TF, TC> flow() {
    while (true) {
      for (int i = 0; i < n; ++i) {
        dis[i] = INF;
        inq[i] = 0;
      }
      dis[s] = 0;
      queue<int> que;
      que.emplace(s);
      while (not que.empty()) {
        int u = que.front();
        que.pop();
        inq[u] = 0;
        for (int i = 0; i < E[u].size(); ++i) {
          int v = E[u][i].v;
          TC w = E[u][i].c;
          if (E[u][i].f > 0 and dis[v] > dis[u] + w) {
            pre[v] = u;
            pre_E[v] = i;
            dis[v] = dis[u] + w;
            if (not inq[v]) {
              inq[v] = 1;
              que.emplace(v);
            }
          }
        }
      }
      if (dis[t] == INF) break;
      TF tf = INF;
      for (int v = t, u, l; v != s; v = u) {
        u = pre[v];
        l = pre_E[v];
        tf = min(tf, E[u][l].f);
      }
      for (int v = t, u, l; v != s; v = u) {
        u = pre[v];
        l = pre_E[v];
        E[u][l].f -= tf;
        E[v][E[u][l].r].f += tf;
      }
      cost += tf * dis[t];
      fl += tf;
    }
    return {fl, cost};
  }
};
```

4.2 Dinic

```
template <class T>
struct Dinic {
  static const int MAXV = 10000;
  static const T INF = 0x3f3f3f3f;
  struct Edge {
    int v;
    T f;
    int re;
    Edge(int _v, T _f, int _re) : v(_v), f(_f), re(_re)
        {}
  };
  int n, s, t, level[MAXV];
  vector<Edge> E[MAXV];
  int now[MAXV];
  Dinic(int _n, int _s, int _t) : n(_n), s(_s), t(_t)
      {}
  void add_edge(int u, int v, T f, bool bidirectional =
      false) {
    E[u].emplace_back(v, f, E[v].size());
    E[v].emplace_back(u, 0, E[u].size() - 1);
    if (bidirectional) {
      E[v].emplace_back(u, f, E[u].size() - 1);
    }
  }
  bool BFS() {
    memset(level, -1, sizeof(level));
    queue<int> que;
    que.emplace(s);
    level[s] = 0;
    while (not que.empty()) {
      int u = que.front();
      que.pop();
      for (auto it : E[u]) {
        if (it.f > 0 and level[it.v] == -1) {
          level[it.v] = level[u] + 1;
          que.emplace(it.v);
        }
      }
    }
    return level[t] != -1;
  }
  T DFS(int u, T nf) {
    if (u == t) return nf;
    T res = 0;
    while (now[u] < E[u].size()) {
      Edge &it = E[u][now[u]];
      if (it.f > 0 and level[it.v] == level[u] + 1) {
        T tf = DFS(it.v, min(nf, it.f));
        res += tf;
        nf -= tf;
        it.f -= tf;
        E[it.v][it.re].f += tf;
        if (nf == 0) return res;
      } else
        ++now[u];
    }
    if (not res) level[u] = -1;
    return res;
  }
  T flow(T res = 0) {
    while (BFS()) {
      T temp;
      memset(now, 0, sizeof(now));
      while (temp = DFS(s, INF)) {
        res += temp;
        res = min(res, INF);
      }
    }
    return res;
  }
};
```

5 Geometry

5.1 Line and points

```
namespace kika {
  using cod = complex<double>;

  const double EPS = 1e-9;
  const double PI = acos(-1);
```

```
int dcmp(double x) {
  if (abs(x) < EPS) return 0;
  return x > 0 ? 1 : -1;
}

bool less(cod a, cod b) {
  return real(a) < real(b) || real(a) == real(b) &&
      imag(a) < imag(b);
}

bool more(cod a, cod b) {
  return real(a) > real(b) || real(a) == real(b) &&
      imag(a) > imag(b);
}

double dot(cod a, cod b) {
  return real(conj(a) * b);
}

double cross(cod a, cod b) {
  return imag(conj(a) * b);
}

int ori(cod b, cod a, cod c) {
  return dcmp(cross(a - b, c - b));
}

double angle(cod a, cod b) {
  return acos(dot(a, b) / abs(a) / abs(b));
}

double sarea(cod a, cod b, cod c) {
  return cross(b - a, c - a);
}

cod rotate(cod a, double rad) {
  return a * cod(cos(rad), sin(rad));
}

cod normal(cod a) {
  return cod(-imag(a) / abs(a), real(a) / abs(a));
}

cod get_line_intersection(cod p, cod v, cod q, cod w)
     { // p and v are two points that decides a line
  cod u(p - q);
  double t = cross(w, u) / cross(v, w);
  return p + v * t;
}

double distance_to_line(cod p, cod a, cod b) {
  return abs(cross(b - a, p - a) / abs(b - a));
}

double distance_to_segment(cod p, cod a, cod b) {
  if (a == b) return abs(p - a);
  cod v1(b - a), v2(p - a), v3(p - b);
  if (dcmp(dot(v1, v2)) < 0) return abs(v2);
  else if (dcmp(dot(v1, v3)) > 0) return abs(v3);
  return abs(cross(v1, v2)) / abs(v1);
}

cod get_line_projection(cod p, cod a, cod b) {
  cod v(b - a);
  return a + dot(v, p - a) / dot(v, v) * v;
}

bool segment_proper_intersection(cod a1, cod a2, cod
    b1, cod b2) {
  double c1 = cross(a2 - a1, b1 - a1), c2 = cross(a2
      - a1, b2 - a1);
  double c3 = cross(b2 - b1, a1 - b1), c4 = cross(b2
      - b1, a2 - b1);
  return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(
      c4) < 0;
}

double polygon_area(vector<cod> p) {
  double area = 0;
  for (int i = 1; i < int(p.size()) - 1; ++i) {
    area += cross(p[i] - p[0], p[i + 1] - p[0]);
  }
  return area / 2;
}

bool is_point_on_segment(cod p, cod a1, cod a2) {
  return dcmp(cross(a1 - p, a2 - p)) == 0 && dcmp(dot
      (a1 - p, a2 - p)) < 0;
}

int is_point_in_polygon(cod p, vector<cod> gon) {
  int wn = 0;
  int n = gon.size();
  for (int i = 0; i < n; ++i) {
    if (is_point_on_segment(p, gon[i], gon[(i + 1) %
        n])) return -1;
    int k = dcmp(cross(gon[(i + 1) % n] - gon[i], p -
         gon[i]));
    int d1 = dcmp(imag(gon[i]) - imag(p));
    int d2 = dcmp(imag(gon[(i + 1) % n] - imag(p)));
    wn += k > 0 && d1 <= 0 && d2 > 0;
    wn -= k < 0 && d2 <= 0 && d1 > 0;
  }
  return wn != 0;
}

vector<cod> convex_hull(vector<cod> p) {
  sort(p.begin(), p.end(), less);
  p.erase(unique(p.begin(), p.end()), p.end());
  int n = p.size(), m = 0;
  vector<cod> ch(n + 1);
  for (int i = 0; i < n; ++i) { // note that border
      is cleared
    while (m > 1 && dcmp(cross(ch[m - 1] - ch[m - 2],
         p[i] - ch[m - 2])) <= 0) {
      --m;
    }
    ch[m++] = p[i];
  }
  for (int i = n - 2, k = m; i >= 0; --i) {
    while (m > k && dcmp(cross(ch[m - 1] - ch[m - 2],
         p[i] - ch[m - 2])) <= 0) {
      --m;
    }
    ch[m++] = p[i];
  }
  ch.erase(ch.begin() + m - (n > 1), ch.end());
  return ch;
}
};
```

6 Graph

6.1 2-SAT

```
#include <cstdio>
#include <vector>
#include <stack>
#include <cstring>
using namespace std;

const int N = 2010;
struct two_SAT {
  int n;
  vector<int> G[N], revG[N];
  stack<int> finish;
  bool sol[N], visit[N];
  int cmp[N];
  void init(int _n) {
    n = _n;
    for (int i = 0; i < N; i++) {
      G[i].clear();
      revG[i].clear();
    }
  }
  void add_edge(int u, int v) {
    // 2 * i -> i is True, 2 * i + 1 -> i is False
    G[u].push_back(v);
    G[v^1].push_back(u^1);
    revG[v].push_back(u);
```

```cpp
    revG[u^1].push_back(v^1);
  }
  void dfs(int v) {
    visit[v] = true;
    for ( auto i:G[v] ) {
      if ( !visit[i] ) dfs(i);
    }
    finish.push(v);
  }
  void revdfs(int v, int id) {
    visit[v] = true;
    for ( auto i:revG[v] ) {
      if ( !visit[i] ) revdfs(i,id);
    }
    cmp[v] = id;
  }
  int scc() {
    memset( visit, 0, sizeof(visit) );
    for (int i = 0; i < 2 * n; i++) {
      if ( !visit[i] ) dfs(i);
    }
    int id = 0;
    memset( visit, 0, sizeof(visit) );
    while ( !finish.empty() ) {
      int v = finish.top(); finish.pop();
      if ( visit[v] ) continue;
      revdfs(v,++id);
    }
    return id;
  }
  bool solve() {
    scc();
    for (int i = 0; i < n; i++) {
      if ( cmp[2*i] == cmp[2*i+1] ) return 0;
      sol[i] = ( cmp[2*i] > cmp[2*i+1] );
    }
    return 1;
  }
} sat;

int main() {
  // ( a or not b ) and ( b or c ) and ( not c or not a )
  sat.init(3);
  sat.add_edge( 2*0+1, 2*1+1 );
  sat.add_edge( 2*1+1, 2*2+0 );
  sat.add_edge( 2*2+0, 2*0+1 );
  printf("%d\n", sat.solve() );
  return 0;
}
```

6.2 maximal cliques

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

const int N = 60;
typedef long long LL;

struct Bron_Kerbosch {
  int n, res;
  LL edge[N];
  void init(int _n) {
    n = _n;
    for (int i = 0; i <= n; i++) edge[i] = 0;
  }
  void add_edge(int u, int v) {
    if ( u == v ) return;
    edge[u] |= 1LL << v;
    edge[v] |= 1LL << u;
  }
  void go(LL R, LL P, LL X) {
    if ( P == 0 && X == 0 ) {
      res = max( res, __builtin_popcountll(R) ); // notice LL
      return;
    }
    if ( __builtin_popcountll(R) + __builtin_popcountll(P) <= res ) return;
    for (int i = 0; i <= n; i++) {
      LL v = 1LL << i;
      if ( P & v ) {
        go( R | v, P & edge[i], X & edge[i] );
        P &= ~v;
        X |= v;
      }
    }
  }
  int solve() {
    res = 0;
    go( 0LL, ( 1LL << (n+1) ) - 1, 0LL );
    return res;
  }
/*  BronKerbosch1(R, P, X):
      if P and X are both empty:
        report R as a maximal clique
      for each vertex v in P:
        BronKerbosch1(R □ {v}, P □ N(v), X □ N(v))
        P := P \ {v}
        X := X □ {v}
*/
} MaxClique;

int main() {
  MaxClique.init(6);
  MaxClique.add_edge(1,2);
  MaxClique.add_edge(1,5);
  MaxClique.add_edge(2,5);
  MaxClique.add_edge(4,5);
  MaxClique.add_edge(3,2);
  MaxClique.add_edge(4,6);
  MaxClique.add_edge(3,4);
  cout << MaxClique.solve() << "\n";
  return 0;
}
```

6.3 Tarjan SCC

```cpp
#include <cstdio>
#include <vector>
#include <stack>
#include <cstring>
using namespace std;

const int N = 10010;
struct Tarjan {
  int n;
  vector<int> G[N], revG[N];
  stack<int> finish;
  bool visit[N];
  int cmp[N];
  void init(int _n) {
    n = _n;
    for (int i = 0; i <= n; i++) {
      G[i].clear();
      revG[i].clear();
    }
  }
  void add_edge(int u, int v) {
    G[u].push_back(v);
    revG[v].push_back(u);
  }
  void dfs(int v) {
    visit[v] = true;
    for ( auto i:G[v] ) {
      if ( !visit[i] ) dfs(i);
    }
    finish.push(v);
  }
  void revdfs(int v, int id) {
    visit[v] = true;
    for ( auto i:revG[v] ) {
      if ( !visit[i] ) revdfs(i,id);
    }
    cmp[v] = id;
  }
  int solve() {
    memset( visit, 0, sizeof(visit) );
    for (int i = 0; i < n; i++) {
      if ( !visit[i] ) dfs(i);
```

```
    }
    int id = 0;
    memset( visit, 0, sizeof(visit) );
    while ( !finish.empty() ) {
      int v = finish.top(); finish.pop();
      if ( visit[v] ) continue;
      revdfs(v,++id);
    }
    return id;
  }
} scc;

int main() {
  int V, E;
  scanf("%d %d", &V, &E);
  scc.init(V);
  for (int i = 0; i < E; i++) {
    int u, v;
    scanf("%d %d", &u, &v);
    scc.add_edge(u-1,v-1);
  }
  printf("%d\n", scc.solve() );
  return 0;
}
```

7 Number Theory

7.1 basic

```
PLL exd_gcd(LL a, LL b) {
  if (a % b == 0) return {0, 1};
  PLL T = exd_gcd(b, a % b);
  return {T.second, T.first - a / b * T.second};
}
LL mul(LL x, LL y, LL mod) {
  LL ans = 0, m = x, s = 0, sgn = (x > 0) xor (y > 0)?
     -1: 1;
  for (x = abs(x), y = abs(y); y; y >>= 1, m <<= 1, m =
      m >= mod? m - mod: m)
    if (y&1) s += m, s = s >= mod? s - mod: s;
  return s * sgn;
}
LL dangerous_mul(LL a, LL b, LL mod){ // 10 times
    faster than the above in average, but could be
    prone to wrong answer (extreme low prob?)
  return (a * b - (LL)((long double)a * b / mod) * mod)
      % mod;
}
LL powmod(LL x, LL p, LL mod) {
  LL s = 1, m = x % mod;
  for (; p; m = mul(m, m, mod), p >>= 1)
    if (p&1) s = mul(s, m, mod);
  return s;
}
```

7.2 Chinese Remainder Theorem

```
typedef long long LL;
typedef pair<LL, LL> PLL;
PLL exd_gcd(LL a, LL b);
LL CRT(vector<PLL> &eqs) {
  LL prod = 1, ans = 0, ni, ns;
  for (auto P: eqs) prod *= P.second;
  for (auto P: eqs) {
    ni = P.second, ns = prod / ni;
    (ans += ns * P.first % prod * exd_gcd(ni, ns).
        second) %= prod;
  }
  return (ans + prod) % prod;
}
```

7.3 Discrete Log

```
LL discrete_log(LL b, LL p, LL n) {
  map<LL, LL> att;
  LL m = sqrt((double)p) + 1, M = powmod(b, m * (p - 2)
     , p);
  for (LL cur = 1, i = 0; i < m; ++i, cur = cur * b % p
     )
    if (not att.count(cur)) att[cur] = i;
  for (LL cur = 1, i = 0; i * m < p - 1; ++i, cur = cur
      * M % p)
    if (att.count(n * cur % p))
      return (att[cur * n % p] + i * m) % (p - 1);
  return -1;
}
// find x s.t. b**x % p == n with complexity O(sqrt(N))
// return the smallest
// return -1 if ans doesn't exist
```

7.4 Lucas

```
LL fac[100000] = {1};
LL C(LL a, LL b, LL p) {
  for (int i = 1; i <= p; ++i) fac[i] = fac[i - 1] * i
      % p;
  LL ans = 1;
  for (;a; a /= p, b /= p) {
    LL A = a % p, B = b % p;
    if (A < B) return 0;
    (ans += fac[A] * powmod(fac[B] * fac[A - B] % p, p
        - 2, p) % p) %= p;
  }
  return ans;
}
```

7.5 Meissel-Lehmer PI

```
LL PI(LL m);
const int MAXM = 1000, MAXN = 650, UPBD = 1000000;
// 650 ~ PI(cbrt(1e11))
LL pi[UPBD] = {0}, phi[MAXM][MAXN];
vector<LL> primes;
void init() {
  fill(pi + 2, pi + UPBD, 1);
  for (LL p = 2; p < UPBD; ++p)
    if (pi[p]) {
      for (LL N = p * p; N < UPBD; N += p)
        pi[N] = 0;
      primes.push_back(p);
    }
  for (int i = 1; i < UPBD; ++i) pi[i] += pi[i - 1];
  for (int i = 0; i < MAXM; ++i)
    phi[i][0] = i;
  for (int i = 1; i < MAXM; ++i)
    for (int j = 1; j < MAXN; ++j)
      phi[i][j] = phi[i][j - 1] - phi[i / primes[j -
          1]][j - 1];
}
LL P_2(LL m, LL n) {
  LL ans = 0;
  for (LL i = n; primes[i] * primes[i] <= m and i <
      primes.size(); ++i)
    ans += PI(m / primes[i]) - i;
  return ans;
}
LL PHI(LL m, LL n) {
  if (m < MAXM and n < MAXN) return phi[m][n];
  if (n == 0) return m;
  LL p = primes[n - 1];
  if (m < UPBD) {
    if (m <= p) return 1;
    if (m <= p * p * p) return pi[m] - n + 1 + P_2(m, n
        );
  }
  return PHI(m, n - 1) - PHI(m / p, n - 1);
}
LL PI(LL m) {
  if (m < UPBD) return pi[m];
  LL y = cbrt(m) + 10, n = pi[y];
  return PHI(m, n) + n - 1 - P_2(m, n);
```

```
}
```

7.6 Miller Rabin with Pollard rho

```
// Miller_Rabin
LL abs(LL a) {return a > 0? a: -a;}
bool witness(LL a, LL n, LL u, int t) {
  LL x = modpow(a, u, n), nx;
  for (int i = 0; i < t; ++i, x = nx){
    nx = mul(x, x, n);
    if (nx == 1 and x != 1 and x != n - 1) return 1;
  }
  return x != 1;
}
const LL wits[7] = {2, 325, 9375, 28178, 450775,
    9780504, 1795265022};
bool miller_rabin(LL n, int s = 7) {
  if (n < 2) return 0;
  if (n&1^1) return n == 2;
  LL u = n - 1, t = 0, a; // n == (u << t) + 1
  while (u&1^1) u >>= 1, ++t;
  while (s--)
    if (a = wits[s] % n and witness(a, n, u, t)) return
         0;
  return 1;
}
// Pollard_rho
LL f(LL x, LL n) {
  return mul(x, x, n) + 1;
}
LL pollard_rho(LL n) {
  if (n&1^1) return 2;
  while (true) {
    LL x = rand() % (n - 1) + 1, y = 2, d = 1;
    for (int sz = 2; d == 1; y = x, sz <<= 1)
      for (int i = 0; i < sz and d <= 1; ++i)
        x = f(x, n), d = __gcd(abs(x - y), n);
    if (d and n - d) return d;
  }
}
```

7.7 Primitive Root

```
vector<LL> factor(LL N) {
  vector<LL> ans;
  for (LL p = 2, n = N; p * p <= n; ++p)
    if (N % p == 0) {
      ans.push_back(p);
      while (N % p == 0) N /= p;
    }
  if (N != 1) ans.push_back(N);
  return ans;
}
LL find_root(LL p) {
  LL ans = 1;
  for (auto q: factor(p - 1)) {
    LL a = rand() % (p - 1) + 1, b = (p - 1) / q;
    while (powmod(a, b, p) == 1) a = rand() % (p - 1) +
         1;
    while (b % q == 0) b /= q;
    ans = mul(ans, powmod(a, b, p), p);
  }
  return ans;
}
bool is_root(LL a, LL p) {
  for (auto q: factor(p - 1))
    if (powmod(a, (p - 1) / q, p) == 1)
      return false;
  return true;
}
```