Contents

1 Setting

1.1 Header

```
#include<bits/stdc++.h>

using namespace std;
typedef long long ll;
typedef unsigned long long ull;
typedef pair<int, int> Pi;
typedef pair<ll,ll> Pll;

#define Fi first
#define Se second
#define pb(x) push_back(x)
#define sz(x) (int)x.size()
#define rep(i, n) for(int i=0;i<n;i++)
#define repp(i, n) for(int i=1;i<=n;i++)
#define all(x) x.begin(), x.end()

#define INF 987654321
#define IINF 654321987654321LL
```

1.2 vimrc

```
syntax on
set nu ai ci si nobk et ar ru nocp hls
set bs=2 ts=4 sw=4 sts=4
set cb=unnamed
set mouse=an
command PS vsp %:r.in|sp %:r.out|vert res 30|wa
command RIO wall|!g++ -O2 -std=c++14 -Wall -lm %:r.cpp && ./a.out < %:r.in > %:r
  .out
command RI  wall|!g++ -O2 -std=c++14 -Wall -lm %:r.cpp && ./a.out < %:r.in
```

1.3 Sublime text

```
{
    "shell_cmd": "g++ -O2 -std=c++11 \"${file}\" -o \"${file_path}/${
      file_base_name}\" && \"${file_path}/${file_base_name}\" < input.txt",
    "working_dir": "${file_path}",
    "selector": "source.c++",
}
```

2 String

2.1 KMP

```
vector<int> preprocess(string p){
    int m = p.size();
    vector<int> fail(m);
    fail[0] = 0; int j = 0;
    for(int i=1;i<m;i++){
        while(j>0&&p[i]!=p[j]) j = fail[j-1];
        if( p[i] == p[j] ){
            fail[i] = j+1; j++;
        }else{
            fail[i] = 0;
        }
    }
    return fail;
}

vector<int> kmp(string s, string p){
    auto fail = preprocess(p);
    vector<int> ans; int n = s.size(), m = p.size();
    int j = 0;
    for(int i=0;i<n;i++){
        while(j>0 && s[i]!=p[j]) j = fail[j-1];
        if( s[i] == p[j] ){
            if( j == m-1 ){
                ans.pb(i-m+1); j = fail[j];
            }else{
                j++;
            }
        }
    }
    return ans;
}
```

2.2 Aho Chorasick

```
struct AhoCorasick{
    struct Node{
        int fail;
        vector<int> output;
        int children[26];

        Node(){
            for(int i=0;i<26;i++) children[i] = -1;
            fail = -1;
        }
    };

    vector<Node> trie;
    int new_node(){
        Node x;
        trie.push_back(x);
        return (int)trie.size()-1;
    }

    void add(int node, string &s, int idx, int string_num){
        //cout << node << " " << idx << endl;
        if( idx == s.size() ){
            trie[node].output.push_back(string_num);
            return;
        }
        int c = s[idx] - 'a';
        if( trie[node].children[c] == -1 ){
            int next = new_node();
            trie[node].children[c] = next;
        }

        add(trie[node].children[c], s, idx+1, string_num);
    }

    void build(vector<string> v){
        int root = new_node();
        for(int i=0;i<v.size();i++){
            add(root,v[i],0,i);
        }

        queue<int> q;
        q.push(root); trie[root].fail = root;
        while( !q.empty() ){
            int cur = q.front(); q.pop();
            for(int i=0;i<26;i++){
                int next = trie[cur].children[i];
                if( next == -1 ) continue;

                // build fail
                if( cur == root ){
                    trie[next].fail = root;
                }
                else{
                    int x = trie[cur].fail;
                    while( x != root && trie[x].children[i] == -1 ) x = trie[x].
                      fail;
                    if( trie[x].children[i] != -1 ) x = trie[x].children[i];
                    trie[next].fail = x;
                }
                // build output
                int f = trie[next].fail;
                for(auto e : trie[f].output) trie[next].output.push_back(e);
                q.push(next);
            }
        }
    }

    vector<Pi> find(string s){
        int n = (int) s.size();
        int cur = 0, root = 0;
        vector<Pi> ans;
        for(int i=0;i<n;i++){
            int c = s[i]-'a';
            while( cur != root && trie[cur].children[c] == -1 ) cur = trie[cur].
              fail;
            if( trie[cur].children[c] != -1 ) cur = trie[cur].children[c];

            for(auto e : trie[cur].output){
                ans.push_back({e,i});
            }

        }
        return ans;
    }
};
```

2.3 Suffix array

```
// Make sure to add !, #, $, %, & at the end of input string
class SuffixArray{
public:
    int n;
    string s;
    vector<int> rank, temprank, sa, tempsa, c;
    vector<int> lcp;
    SuffixArray(string _s){
        n = _s.size(); s = _s;
        rank.resize(n); temprank.resize(n); sa.resize(n); tempsa.resize(n);
        lcp.resize(n);
        constructSA();
        constructLCP();
    }

    void countingSort(int k){
        int sum = 0, maxi = max(270, n); //ASCII 256
        c.clear(); c.resize(maxi+10);
        for(auto& e : c ) e = 0;
        for(int i=0; i<n; i++) c[ i+k<n ? rank[i+k] : 0 ] ++;
        for(int i=0; i<maxi; i++){
            int t = c[i]; c[i] = sum; sum += t;
        }
        for(int i=0; i<n; i++) tempsa[ c[ sa[i]+k < n ? rank[sa[i]+k] : 0 ] ++ ]
           = sa[i];
        for(int i=0; i<n; i++) sa[i] = tempsa[i];
    }


    void constructSA(){
        for(int i=0; i<n; i++) rank[i] = s[i];
        for(int i=0; i<n; i++) sa[i] = i;
        for(int k=1; k<n; k<<=1){
            countingSort(k);
            countingSort(0);
            int r = 0;
            temprank[sa[0]] = 0;
            for(int i=1; i<n; i++){
                temprank[sa[i]] = (rank[sa[i]] == rank[sa[i-1]] && rank[sa[i]+k]
                   == rank[sa[i-1]+k] ) ? r : ++r;
            }
            for(int i=0; i<n; i++) rank[i] = temprank[i];
            if( rank[sa[n-1]] == n-1 ) break;
        }
    }


    // lcp Implementation from
    // http://m.blog.naver.com/dark__nebula/220419358547
    void constructLCP(){
        int h = 0;
        for(int i=0;i<n;i++){
            if( rank[i] ){
                int j = sa[rank[i]-1];
                while( s[i+h] == s[j+h] ) h++;
                lcp[rank[i]] = h;
            }
            if( h > 0 ) h--;
        }
    }

};
```

2.4 Manacher's algorithm

```
// finds radius of longest palindrome centered at s[i]
// If you also want to find even-length paindromes, use dummy characters
// baab -> #b#a#a#b#
vector<int> ManacherAlgorithm(string s){
    int n = (int) s.size();
    int p = -1, r = -1;
    vector<int> A(n);
    for(int i=0;i<n;i++){

        if( r < i ){
            A[i] = 0;
            int j = 0;
            while( i + A[i] < n && i - A[i] >= 0 && s[ i+A[i] ] == s[ i-A[i] ] )
               A[i]++;
            A[i]--;
        }
        else{
            A[i] = min( A[2*p - i] , r-i );
            while( i + A[i] < n && i - A[i] >= 0 && s[ i+A[i] ] == s[ i-A[i] ] )
               A[i]++;
            A[i]--;
        }

        // update r
        if( r < i + A[i] ){
            r = i + A[i];
            p = i;
        }
    }
    return A;
}
```

2.5 Z algorithm

```
// Calculates LCP[i] for all 0 <= i < n
vector<int> Zalgorithm(string s){
    int l=0, r=0;
    int n = (int) s.size();
    vector<int> Z(n);
    Z[0] = n;
    for(int i=1; i<n; i++){
        // reset and calculate again
        if( i > r ){
            l = r = i;
            while( r<n && s[r] == s[r-l] ) r++;
            r--;
            Z[i] = r-l+1;
        }

        // extend [l,r]
        else{
            int k = i-l;
            // not enough matching at position k
            if( Z[k] < r-i+1 ) Z[i] = Z[k];
            // enough matching. extend [l,r]
            else{
                l = i;
                while( r<n && s[r] == s[r-l] ) r++;
                r--;
                Z[i] = r-l+1;
            }
        }

    }
    return Z;
};
```

3 Graph & Flow

3.1 BCC

```
int N,M;
int timer = 0;
vector<int> E[300500];
int vis[300500], low[300500];

// dfs1 is to fill vis(discover time) and low array
int dfs1(int x, int pa){
    vis[x] = ++timer;
    low[x] = vis[x];
    for(auto e : E[x])if(e!=pa){
        if( vis[e] ){
            low[x] = min(low[x], vis[e]);
        }
        else{
            dfs1(e,x);
            low[x] = min(low[x], low[e]);
        }
    }
    return low[x] ;
}

int color = 0;
vector<int> colors[300500], E2[300500];
int vis2[300500];

// dfs2 is to color every nodes
// Store node's colors into colors array
// Store new edges into E2
void dfs2(int x, int pa, int c){
    colors[x].pb(c);
    vis2[x] = 1;
    for(auto e : E[x])if(!vis2[e]){
        // x-e is an articulation edge
        if( low[e] > vis[x] ){
            ++color;
            colors[x].pb(color);
            E2[c].pb(color); E2[color].pb(c);
            dfs2(e,x,color);
        }
        // x-e is not an articulation edge
        else dfs2(e,x,c);

    }
}

int main(){
    geti(N,M);
    repp(i,M){
        int a, b; geti(a,b);
        E[a].pb(b); E[b].pb(a);
    }
    // fill vis & low
    dfs1(1,-1);
    // find out articulation edge and color of nodes
    color = 1;
    dfs2(1,-1,color);
}
```

3.2 Maximum Clique

```
ll G[40]; // 0-index
void get_clique(int R = 0, ll P = (1ll<<N)-1, ll X = 0){
  if((P|X) == 0){
    cur = max(cur, R);
    return;
  }
  int u = __builtin_ctzll(P|X);
  ll c = P&~G[u];
  while(c){
    int v = __builtin_ctzll(c);
    get_clique(R + 1, P&G[v], X&G[v]);
    P ^= 1ll << v;
    X |= 1ll << v;
    c ^= 1ll << v;
  }
}
```

3.3 Hopcroft Karp

```
namespace Matching{
//matching [1...n] <-> [1...m]
const int MX = 40040, MY = 40040;
vector <int> E[MX];
int xy[MX], yx[MY];
int n, m;

void addE(int x, int y) { E[x].pb(y); }
void setnm(int sn, int sm) { n = sn; m = sm; }

int tdis[MX], que[MX], *dis = tdis + 1;
int bfs() {
  int *fr = que, *re = que;
  for(int i=1;i<=n;i++) {
    if(xy[i] == -1) *fr++ = i, dis[i] = 0;
    else dis[i] = -1;
  }
  dis[-1] = -1;
  while(fr != re) {
    int t = *re++;
    if(t == -1) return 1;
    for(int e : E[t]) {
      if(dis[yx[e]] == -1) dis[yx[e]] = dis[t] + 1, *fr++ = yx[e];
    }
  }
  return 0;
}

int dfs(int x) {
  for(int e : E[x]) {
    if(yx[e] == -1 || (dis[yx[e]] == dis[x] + 1 && dfs(yx[e]))) {
      xy[x] = e;
      yx[e] = x;
      return 1;
    }
  }
  dis[x] = -1;
  return 0;
}

int Do() {
  memset(xy, -1, sizeof xy);
  memset(yx, -1, sizeof yx);

  int ans = 0;
  while(bfs()) {
    for(int i=1;i<=n;i++) if(xy[i] == -1 && dfs(i)) ++ans;
  }
  return ans;
}
}


void solve(){
  int n, m;
  scanf("%d%d", &n, &m);
  Matching::setnm(n, m);
  for(int i=1;i<=n;i++) {
    int x; scanf("%d", &x);
    while(x--) {
      int y; scanf("%d", &y);
      Matching::addE(i, y);
    }
  }
  printf("%d\n", Matching::Do());
}
```

3.4 Dinic

```
struct MaxFlowDinic{
    struct Edge{
        // next, inv, residual
        int to, inv; ll res;
    };

    int n;
    vector<vector<Edge>> graph;

    vector<int> lev,work;

    void init(int x){
        n = x+10;
        graph.resize(x+10);
        lev.resize(n); work.resize(n);
    }

    void make_edge(int s, int e, ll cap, ll caprev = 0){
        Edge forward = {e, (int)graph[e].size(), cap};
        Edge backward = {s, (int)graph[s].size(), caprev};
        graph[s].push_back(forward);
        graph[e].push_back(backward);
    }

    bool bfs(int source, int sink){
        queue<int> q;
        for(auto& e : lev) e = -1;
        lev[source] = 0; q.push(source);
        while(!q.empty()){
            int cur = q.front(); q.pop();
            for(auto e : graph[cur]){
                if(lev[e.to]==-1 && e.res > 0){
                    lev[e.to] = lev[cur]+1;
                    q.push(e.to);
                }
            }
        }
        return lev[sink] != -1;
    }

    ll dfs(int cur, int sink, ll flow){
        if( cur == sink ) return flow;
        for(int &i = work[cur]; i < (int)graph[cur].size(); i++){
            Edge &e =  graph[cur][i];
            if( e.res == 0 || lev[e.to] != lev[cur]+1 ) continue;
            ll df = dfs(e.to, sink, min(flow, e.res) );
            if( df > 0 ){
                e.res -= df;
                graph[e.to][e.inv].res += df;
                return df;
            }
        }
        return 0;
    }


    ll solve( int source, int sink ){
        ll ans = 0;
        while( bfs(source, sink) ){
            for(auto& e : work) e = 0;
            while( true ){
                ll flow = dfs(source,sink,54321987654321LL);
                if( flow == 0 ) break;
                ans += flow;
            }
        }
        return ans;
    }

};
```

3.5 MCMF

```
struct MCMF{
    struct edge{
        int to, inv, cap, flow, cost;
        int res(){
            return cap - flow;
        }
    };

    vector<vector<edge>> graph;
    vector<int> pv, pe;
    vector<int> dist, inq;

    void init(int x){
        graph.resize(x+10);
        for(auto& e : graph) e.resize(x+10);
        pv.resize(x+10); pe.resize(x+10);
        dist.resize(x+10);
        inq.resize(x+10);
    }

    void make_edge(int from, int to, int cap, int cost){
        //printf("%d -> %d | cost = %d\n",from,to,cost);
        edge forward = {to, (int)graph[to].size(), cap, 0, cost};
        edge backward = {from, (int)graph[from].size(), 0, 0, -cost};
        graph[from].push_back(forward);
        graph[to].push_back(backward);
    }

    int solve(int source, int sink){
        int ans = 0;
        int totalflow = 0;
        while(true){
            for(auto& e : dist) e = INF;
            for(auto& e : inq) e = 0;
            queue<int> q;
            q.push(source); inq[source] = 1;
            dist[source] = 0;

            while(!q.empty()){
                int cur = q.front(); q.pop();
                inq[cur] = 0;
                for(int i=0;i<(int)graph[cur].size();i++){
                    auto& e = graph[cur][i];
                    if( e.res() > 0 && dist[e.to] > dist[cur] + e.cost ){
                        dist[e.to] = dist[cur] + e.cost;
                        pv[e.to] = cur; pe[e.to] = i;
                        if( inq[e.to] == 0 ){
                            q.push(e.to); inq[e.to] = 1;
                        }
                    }
                }
            }

            if( dist[sink] == INF ) break;

            // add this limit when we don't require maxflow
            //if( dist[sink] > 0 ) break;

            int mnflow = INF;
            for( int v = sink; v != source; v = pv[v] ){
                mnflow = min( mnflow, graph[pv[v]][pe[v]].res() );
            }

            for( int v = sink; v != source; v = pv[v] ){
                int tmp = graph[pv[v]][pe[v]].inv;
                graph[pv[v]][pe[v]].flow += mnflow;
                graph[v][tmp].flow -= mnflow;
            }
            totalflow += mnflow;
            ans += dist[sink] * mnflow;
        }
        return ans;
    }

};
```

3.6 Stoer Wagner

```
// Stoer-Wagner algorithm
struct mincut {
    int n;
    vector<vector<int>> graph;

    void init(int nn) {
        n = nn;
        graph.resize(n, vector<int>(n, 0));
    }

    void addEdge(int u, int v, int w) {
        graph[u][v] += w;
        graph[v][u] += w;
    }

    pair<int, vector<int>> findMincut() {
        vector<vector<int>> weight = graph;
        vector<bool> used(n, 0);
        vector<int> best_cut;
        int best_weight = -1;

        vector<vector<int>> group(n);
        for(int i = 0; i < n; i++)
            group[i].push_back(i);


        for(int phase = n-1; phase >= 0; phase--) {
            int start = 0;
            vector<int> w = weight[start];
            vector<bool> inSet = used;
            inSet[start] = true;
            int prev, last = start;

            for(int i = 0; i < phase; i++) {
                prev = last;
                last = -1;
                for(int j = 0; j < n; j++)
                    if(!inSet[j] && (last == -1 || w[j] > w[last])) last = j;

                if(i < phase-1) {
                    inSet[last] = true;
                    for(int j = 0; j < n; j++)
                        w[j] += weight[last][j];
                } else {  // last step - merge two nodes: prev & last
                    for(int j = 0; j < n; j++) {
                        weight[prev][j] += weight[last][j];
                        weight[j][prev] = weight[prev][j];
                    }
                    used[last] = true;
                    group[prev].insert(group[prev].end(), group[last].begin(),
                      group[last].end());
                    if(best_weight == -1 || w[last] < best_weight) {
                        best_weight = w[last];
                        best_cut = group[last];
                    }
                }
            }
        }
        return make_pair(best_weight, best_cut);
    }
};
```

3.7 Arborescence

```
namespace Arborescence{
  const int MX = 510, INF = 1e9;
  int e[MX][MX], lst[MX][MX];
  vector<int> v[MX], rev[MX], order;
  int was[MX], vst[MX], ans[MX], p[MX];
  vector<pii> G[MX];

  int find(int x){ return p[x] == x? x : p[x] = find(p[x]); }
  void set_graph(int ee[MX][MX]){ memcpy(e, ee, sizeof e); }

  void go(int x) {
    if(vst[x]) return;
    vst[x] = 1;
    for (int to : v[x]) go(to);
    order.pb(x);
  }

  void col(int x, int o) {
    if (was[x]) return;
    was[x] = o;
    for (int to : rev[x]) col(to, o);
  }

  int run(int n, int root) {
    int ret = 0, done = 0;
    for(int i = 1; i <= n; i++) p[i] = i;
    memset(lst, 0, sizeof lst);
    for(int tt = 1;;tt++) {
      memset(was, 0, sizeof was);
      memset(vst, 0, sizeof vst);
      for (int i = 1; i <= n; i++) {
        v[i].clear();
        rev[i].clear();
      }
      order.clear();

      int mn[MX] = {};
      for(int i = 1; i <= n; i++) mn[i] = INF;
      for (int i = 1; i <= n; i++) if (find(i) != find(root))
        for (int j = 1; j <= n; j++) if(find(i) != find(j))
          mn[find(i)] = min(mn[find(i)], e[j][i]);
      for (int i = 1; i <= n; i++) if (find(i) != find(root)) {
        if(find(i) == i) ret += mn[i];
        for (int j = 1; j <= n; j++) if(find(i) != find(j)) e[j][i] -= mn[find(i
          )];
      }
      for (int i = 1; i <= n; i++) for (int j = 1; j <= n; j++){
        int a = find(i), b = find(j);
        if (a != b && e[i][j] == 0) {
          lst[i][j] = tt;
          v[a].pb(b);
          rev[b].pb(a);
        }
      }
      if (done) break;
      for (int i = 1; i <= n; i++) if (!vst[i]) go(i);
      reverse(order.begin(), order.end());
      for(int u : order) if (!was[u]) col(u, u);
      done = 1;
      for(int i = 1; i <= n; i++) if(was[i] != i) done = 0, p[i] = was[i];
    }
    priority_queue<t3, vector<t3>, greater<t3>> Q;

    memset(ans, -1, sizeof ans);
    ans[root] = 0;
    for(int i = 1; i <= n; i++) for(int j = 1; j <= n; j++)
      if(e[i][j] == 0) G[i].emplace_back(lst[i][j], j);
    for(pii c : G[root]) Q.emplace(c.first, root, c.second);
    while(Q.size()){
      int a, b; tie(ignore, a, b) = Q.top(); Q.pop();
      if(ans[b] != -1) continue;
      ans[b] = a;
      for(pii c : G[b]) Q.emplace(c.first, b, c.second);
    }
    return ret;
  }
};
;
```

3.8 Dominator Tree

```
#include<vector>
using namespace std;
#define pb(x) push_back(x)
namespace dtree{
const int MAXN = 100010;
vector <int> E[MAXN];
vector <int> RE[MAXN], rdom[MAXN];
int S[MAXN], RS[MAXN], cs;
int par[MAXN], val[MAXN];
int sdom[MAXN], rp[MAXN];
int dom[MAXN];

int Find(int x, int c = 0) {
  if(par[x] == x) return c ? -1 : x;
  int p = Find(par[x], 1);
  if(p == -1) return c ? par[x] : val[x];
  if(sdom[val[x]] > sdom[val[par[x]]]) val[x] = val[par[x]];
  par[x] = p;
  return c ? p : val[x];
}

void Union(int x, int y) {
  par[x] = y;
}

void dfs(int x) {
  RS[ S[x] = ++cs ] = x;
  par[cs] = sdom[cs] = val[cs] = cs;
  for(int e : E[x]) {
    if(S[e] == 0) dfs(e), rp[S[e]] = S[x];
    RE[S[e]].pb(S[x]);
  }
}

int Do(int s, int *up) {
  dfs(s);
  for(int i=cs;i;i--) {
    for(int e : RE[i]) sdom[i] = min(sdom[i], sdom[Find(e)]);
    if(i > 1) rdom[sdom[i]].pb(i);
    for(int e : rdom[i]) {
      int p = Find(e);
      if(sdom[p] == i) dom[e] = i;
      else dom[e] = p;
    }
    if(i > 1) Union(i, rp[i]);
  }
  for(int i=2;i<=cs;i++) if(sdom[i] != dom[i]) dom[i] = dom[dom[i]];
  for(int i=2;i<=cs;i++) {
    up[RS[i]] = RS[dom[i]];
  }
  return cs;
}

void addE(int x, int y) { E[x].pb(y); }
}
```

3.9 Vizing's Algorithm

```
// Color every edge in G with (max degree)+1 colors.
// Edges with shared vertex must have distinct colors.

typedef pair<int,int> pii;
```

```
const int MX = 2505;
int C[MX][MX] = {}, G[MX][MX] = {};

void solve(vector<pii> &E, int N, int M){
  int X[MX] = {}, a, b;

  auto update = [&](int u){ for(X[u] = 1; C[u][X[u]]; X[u]++); };
  auto color = [&](int u, int v, int c){
    int p = G[u][v];
    G[u][v] = G[v][u] = c;
    C[u][c] = v; C[v][c] = u;
    C[u][p] = C[v][p] = 0;
    if( p ) X[u] = X[v] = p;
    else update(u), update(v);
    return p; };
  auto flip = [&](int u, int c1, int c2){
    int p = C[u][c1], q = C[u][c2];
    swap(C[u][c1], C[u][c2]);
    if( p ) G[u][p] = G[p][u] = c2;
    if( !C[u][c1] ) X[u] = c1;
    if( !C[u][c2] ) X[u] = c2;
    return p; };

  for(int i = 1; i <= N; i++) X[i] = 1;
  for(int t = 0; t < E.size(); t++){
    int u = E[t].first, v0 = E[t].second, v = v0, c0 = X[u], c = c0, d;
    vector<pii> L;
    int vst[MX] = {};
    while(!G[u][v0]){
      L.emplace_back(v, d = X[v]);
      if(!C[v][c]) for(a = (int)L.size()-1; a >= 0; a--) c = color(u, L[a].first, c);
      else if(!C[u][d])for(a=(int)L.size()-1;a>=0;a--)color(u,L[a].first,L[a].second);
      else if( vst[d] ) break;
      else vst[d] = 1, v = C[u][d];
    }
    if( !G[u][v0] ){
      for(;v; v = flip(v, c, d), swap(c, d));
      if(C[u][c0]){
        for(a = (int)L.size()-2; a >= 0 && L[a].second != c; a--);
        for(; a >= 0; a--) color(u, L[a].first, L[a].second);
      } else t--;
    }
}
}
```

3.10 LR-flow

```
G has a feasible (s,t)-flow iff G' has a saturating (s',t')-flow
in G' total capacity out of s' and into t' are both D (sum of demands)
saturating flow : flow with value exactly D.

1. Make new source, new sink (s', t')

2. for every v:
c'(s'->v) = sum{ d(u->v) } (give demands into v)
c'(v->t') = sum{ d(v->w) } (take demands out of v)

3. for every u->v:
c'(u->v) = c(u->v) - d(u->v)  (difference of cap, demand)

3. make t->s cap:INF
```

4 Query

4.1 Splay Tree

```
const int N_ = 2e5;
const int inf = ~0u>>1;

struct node{
  inline void pushdown()
  {
    if( rev ){
      if( link[0] ) link[0]->rev ^= 1;
      if( link[1] ) link[1]->rev ^= 1;
      swap( link[0], link[1] );
      rev = 0;
    }
    if( add ){
      if( link[0] ) link[0]->add += add, link[0]->mn += add, link[0]->val += add;
      if( link[1] ) link[1]->add += add, link[1]->mn += add, link[1]->val += add;
      add = 0;
    }
  }

  inline void pushup()
  {
    cnt = (link[0]? link[0]->cnt:0) + (link[1]? link[1]->cnt:0) + 1;
    mn = min( val, min(link[0]?link[0]->mn:inf, link[1]?link[1]->mn:inf));
  }

  int cnt, add, mn, val; //cnt: number of nodes
  bool rev;
  node *link[2], *par;
};

struct splaytree{
  node N[ N_ ];
  node* root;
  int sz;

  node* operator[](int idx){ return N + idx; }

  void clear(int s){
    sz = 0;
    for(int i=0;i<=s+2;i++){
      N[i].link[0] = N[i].link[1] = N[i].par = 0, N[i].cnt = 1;
      N[i].rev = false;
    }
// dummy nodes can remove many null-pointer exceptions
    root = N+s+1; root->cnt = 2;
    N[s+2].par = N+s+1; N[s+1].link[1] = N+s+2;
  }

  inline int dir(node *x){ return x->par->link[0] != x; }
  inline int cnt(node* p){ return p? p->cnt: 0; }

  void rotate(node *n) // To
  {
    n->par->pushdown(); n->pushdown();
    node *p = n->par;
    int d = dir(n);
    p->link[d] = n->link[!d];   if( n->link[!d] ) n->link[!d]->par = p;
    n->par = p->par;  if( p->par ) p->par->link[ dir(p) ] = n;
    n->link[!d] = p;  p->par = n;
    p->pushup(); n->pushup();
  }

  void splay(node *x, node *f){
    if( x == f ) return;
    while(x->par != f){
      x->par->pushdown();
      if( x->par->par == f );
      else if(dir(x) == dir(x->par)) rotate(x->par);
      else rotate(x);
      rotate(x);
    }
    x->pushdown();
    if( f == NULL ) root = x;
  }
// 1-index if dummy node exists
  node* kth_splay(int k,node* f)
  {
    node *x = root;
    x->pushdown();
    while( cnt( x->link[0] ) != k ){
      if( cnt( x->link[0] ) < k ){
        if( !x->link[1] ) return x;
        k -= cnt(x->link[0]) + 1, x = x->link[1];
      }
      else x = x->link[0];
      x->pushdown();
    }
    splay( x, f );
    return x;
  }
// 1-index if dummy nodes exist
// recommend: 'dont copy & paste code below.
// be careful if dummy nodes 'dont exist (ex. null-pointer exception)
  void insert(int wi, node *n)
  {
    if( !root ){
      root = n;
      return;
    }
    kth_splay(wi-1, 0);
    kth_splay(wi, root);
    root->link[1]->link[0] = n; n->par = root->link[1];
    root->link[1]->pushup(); root->pushup();
  }

  void Delete(int x){
    kth_splay(x-1,0);
    kth_splay(x+1,root);
    root->link[1]->link[0] = NULL;
    root->link[1]->pushup(); root->pushup();
  }

  void Reverse(int x,int y){
    if( x > y ) return;
    kth_splay(x-1,0);
    kth_splay(y+1,root);
    root->link[1]->link[0]->rev ^= 1;
  }

  void revolve(int x,int y,int T){ // rotate x~y T times
    if( x >= y ) return;
    int l = (y-x+1);
    T = (T%l+l) % l;
    Reverse(x,y-T);
    Reverse(y-T+1,y);
    Reverse(x,y);
  }

  int node_address(int wi)
  {
    node *p = N+wi;
    splay(p, 0);
    return cnt( p->link[0] );
  }

  int min(int x,int y){
    kth_splay(x-1,0);
    kth_splay(y+1,root);
    return root->link[0]->link[1]->mn;
  }
} pre, post;
```

4.2 Link Cut Tree

```
#define _CRT_SECURE_NO_WARNINGS
#include<algorithm>
#include<stdio.h>

using namespace std;
const int N_ = 2e5;
```

```
struct node{
  void pushup(){
    cnt = (link[0]? link[0]->cnt:0) + (link[1]? link[1]->cnt:0) + 1;
    mx = max( max( link[0]? link[0]->mx:0, link[1]? link[1]->mx:0 ), val);
  }

  int cnt, val, mx; //cnt: number of nodes
  node *link[2], *par, *path_parent;
};

struct linkcuttree{
  node N[ N_ ];

  void clear(int s){
    for(int i=0;i<=s;i++)
      N[i].link[0] = N[i].link[1] = N[i].par = N[i].path_parent = 0, N[i].cnt =
        1;
  }

  inline int dir(node *x){ return x->par->link[0] != x; }
  inline int cnt(node *x){ return x?x->cnt:0; }
  inline int mx(node *x){ return x?x->mx:0; }

  void rotate(node *n) // To
  {
    if( !n->par ) return;
    node *p = n->par;
    int d = dir(n);
    n->path_parent = p->path_parent; p->path_parent = NULL;
    p->link[d] = n->link[!d];   if( n->link[!d] ) n->link[!d]->par = p;
    n->par = p->par;  if( p->par ) p->par->link[ dir(p) ] = n;
    n->link[!d] = p;  p->par = n;
    p->pushup(); n->pushup();
  }

  void splay(node *x){
    while( x->par ){
      if( !x->par->par );
      else if(dir(x) == dir(x->par)) rotate(x->par);
      else rotate(x);
      rotate(x);
    }
  }

  void access(node* x)
  {
    splay(x);
    if( x->link[1] ) x->link[1]->path_parent = x, x->link[1]->par = NULL;
    x->link[1] = NULL; x->pushup();
    while( x->path_parent ){
      node *pp = x->path_parent, *r;
      splay(pp);
      r = pp->link[1];
      if( r ) r->par = NULL, r->path_parent = pp;
      pp->link[1] = x; pp->pushup(); x->par = pp;
      x->path_parent = NULL;
      splay(x);
    }
  }

  void cut(int u)
  {
    access(N+u);
    if( N[u].link[0] ) N[u].link[0]->par = NULL;
    N[u].link[0] = NULL; N[u].pushup();
  }

  void link(int u, int v) // u must be root.
  {
    if( u == v ) return;
    access(N+u);
    access(N+v);
    //assert(!N[u].link[0]);
    N[u].link[0] = N+v; N[v].par = N+u; N[u].pushup();
  }

// recommend: ’dont copy & paste code below.
  int read(int u)
  {
    access( N+u );
    return N[u].cnt;
}

  int root(int u)
  {
    access( N+u );
    node* ans = N+u;
    while( ans->link[0] ) ans = ans->link[0];
    splay(ans);
    return ans - N;
  }

  int mx(int u)
  {
    access( N+u );
    return N[u].max;
  }

  bool chk()
  {
    for(int i=0;i<N_;i++){
      if( N[i].cnt == 0 ) return true;
      if( N[i].cnt != cnt(N[i].link[0]) + cnt(N[i].link[1]) + 1) return false;
      if( N[i].mx != max( max( mx(N[i].link[0]), mx(N[i].link[1]) ), N[i].val) )
         return false;
      if( N[i].par && N+i != N[i].par->link[dir(N+i)] ) return false;
      if( N[i].link[0] && N+i != N[i].link[0]->par) return false;
      if( N[i].link[1] && N+i != N[i].link[1]->par) return false;
    }
    return true;
  }
}LCT;
```

4.3 Mo Hilbert Order

```
inline int64_t gilbertOrder(int x, int y, int pow, int rotate) {
        if (pow == 0) {
                return 0;
        }
        int hpow = 1 << (pow-1);
        int seg = (x < hpow) ? (
                (y < hpow) ? 0 : 3
        ) : (
                (y < hpow) ? 1 : 2
        );
        seg = (seg + rotate) & 3;
        const int rotateDelta[4] = {3, 0, 0, 1};
        int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
        int nrot = (rotate + rotateDelta[seg]) & 3;
        int64_t subSquareSize = int64_t(1) << (2*pow - 2);
        int64_t ans = seg * subSquareSize;
        int64_t add = gilbertOrder(nx, ny, pow-1, nrot);
        ans += (seg == 1 || seg == 2) ? add : (subSquareSize - add - 1);
        return ans;
}

struct Query {
        int l, r, idx;
        int64_t ord;

        inline void calcOrder() {
                ord = gilbertOrder(l, r, 21, 0);
        }
};

inline bool operator<(const Query &a, const Query &b) {
        return a.ord < b.ord;
}
```

4.4 Parallel binary search

```
int N,M,K,Q;

vector<Pi> edge[1000500];
int pa[MAXN]; int sz[MAXN];

// each query's answer
Pi ans[MAXN];
// each query's possible answer range for binary search
int low[MAXN], high[MAXN];
// focus[x] : list of query # where it's mid value is x
vector<int> focus[1000500];

int find(int x){
    if( x == pa[x] ) return x;
    return pa[x] = find(pa[x]);
}
int x[MAXN], y[MAXN];

void uni(int a, int b){
    a = find(a); b = find(b);
    if( a == b ) return;
    pa[a] = b;
    sz[b] += sz[a];
}

int main(void){
    //ios::sync_with_stdio(false);
    geti(N,M);
    int C = -1;
    repp(i,M){
        int a,b,c; geti(a,b,c);
        edge[c].push_back({a,b});
        C = max(C, c);
    }

    geti(Q);
    repp(i,Q){
        int a,b;
        geti(a,b); x[i] = a; y[i] = b;
        ans[i] = {INF,-1};
        // Initially, every query has answer in [0,C] range
        low[i] = 0; high[i] = C;
    }

    bool changed = true;
    while( changed ){
        changed = false;

        // Clear variables
        rep(i,C+1) focus[i].clear();
        repp(i,N) pa[i] = i, sz[i] = 1;

        // Put each query into corresponding focus group
        repp(i,Q){
            if( low[i] > high[i] ) continue;
            focus[ (low[i] + high[i])/2 ].push_back(i);
        }

        // for every time 0~C
        for(int k=0;k<=C;k++){
            // perform action of that time
            for(auto e : edge[k]) uni(e.Fi,e.Se);

            // for each focus group
            // determine it's answer & next position
            for(auto e : focus[k]){
                changed = true;
                int a = x[e]; int b = y[e];
                if( find(a) == find(b) ){
                    ans[e].Fi = min(ans[e].Fi, k);
                    ans[e].Se = sz[find(a)];
                    high[e] = k-1;
                }
                else{
                    low[e] = k+1;
                }
            }
        }
    }

    repp(i,Q){
        if( ans[i].Fi == INF ) printf("%d\n",-1);
        else printf("%d %d\n",ans[i].Fi, ans[i].Se);
    }

}
```

4.5 Lazy Propagation 1

```
struct segTree{
    struct Node{
        ll d, lazy;
    };
    vector<Node> data;
    int n;
    void init(int x){
        n = 1; while( n < x ) n *= 2;
        data.resize(n*2+10);
    }
    void propagate(int node, int nodeL, int nodeR){
        if( data[node].lazy == 0 ) return;
        ll len = nodeR - nodeL + 1;
        data[node].d += len*data[node].lazy;
        if( len > 1 ){
            data[node*2].lazy += data[node].lazy;
            data[node*2+1].lazy += data[node].lazy;
        }
        data[node].lazy = 0;
    }

    void update(int l, int r, ll val, int node, int nodeL, int nodeR){
        propagate(node, nodeL, nodeR);
        if( l > nodeR || r < nodeL ) return;
        if( l <= nodeL && nodeR <= r ){
            data[node].lazy += val;
            propagate(node,nodeL,nodeR);
            return;
        }
        update(l,r,val,node*2,nodeL,(nodeL+nodeR)/2);
        update(l,r,val,node*2+1,(nodeL+nodeR)/2+1,nodeR);
        data[node].d = data[node*2].d + data[node*2+1].d;
    }

    ll query(int l, int r, int node, int nodeL, int nodeR){
        propagate(node, nodeL, nodeR);
        if( l > nodeR || r < nodeL ) return 0;
        if( l <= nodeL && nodeR <= r ){
            return data[node].d;
        }
        ll sum = 0;
        sum += query(l,r,node*2,nodeL,(nodeL+nodeR)/2);
        sum += query(l,r,node*2+1,(nodeL+nodeR)/2+1,nodeR);
        return sum;
    }

};
```

5 Geometry

6 Math

6.1 FFT

```
#include <cmath>
#include <complex>
using namespace std;
typedef pair<int,int> pii;
typedef complex<double> base;

void fft(vector<base> &a, bool invert){
    int n = a.size();
    for(int i=1,j=0;i<n;i++){
        int bit = n >> 1;
        for (;j>=bit;bit>>=1)j -= bit;
        j += bit;
        if (i < j) swap(a[i], a[j]);
    }
    for(int len=2;len<=n;len<<=1){
        double ang = 2*acos(-1)/len*(invert?-1:1);
        base wlen(cos(ang),sin(ang));
        for(int i=0;i<n;i+=len){
            base w(1);
            for(int j=0;j<len/2;j++){
                base u = a[i+j], v = a[i+j+len/2]*w;
                a[i+j] = u+v;
                a[i+j+len/2] = u-v;
                w *= wlen;
            }
        }
    }
    if (invert) {
        for(int i=0;i<n;i++) a[i] /= n;
    }
}

void multiply(const vector<int> &a, const vector<int> &b, vector<int> &res){
    vector<base> fa(a.begin(), a.end()), fb(b.begin(),b.end());
    int n = 1;
    while(n < max(a.size(), b.size())) n <<= 1;
    n <<= 1;
    fa.resize(n); fb.resize(n);
    fft(fa,false);fft(fb,false);
    for(int i=0;i<n;i++) fa[i] *= fb[i];
    fft(fa,true);
    res.resize(n);
    for(int i=0;i<n;i++) res[i] = int(fa[i].real() + (fa[i].real() > 0 ? 0.5 :
      -0.5));
}
```

6.2 Kirchhoff Theorem

```
Find number of MST in given graph G.
m[i][j] := -( number of i<->j edges ) (i != j)
m[i][i] := degree of vertex i
(ans) = (det of (n-1)x(n-1) matrix obtained from m with first row&col deleted )

(의m 첫번째행과첫번째열을없앤       (n-1) by (n-1) 의matrix 행렬식)
```

6.3 Simplex

```
/*
LP Duality
tableu 를대각선으로뒤집고음수부호를붙인답       =  -( 원문제의답   )
ex) n = 2, m = 3, a = [[0.5, 2, 1], [1, 2, 4]], b = [24, 60], c = [6, 14, 13]
<=> n = 3, m = 2, a = [[-0.5, -1], [-2, -2], [-1, -4]], b = [-6, -14, -13], c =
  [-24, -60]

n := number of variables
m := number of constraints
a[1~m][1~n] := constraints
b[1~m] := constraints value (b[i] can be negative)
c[1~n] := maximum coefficient
v := results
sol[i] := 등호조건, i 번째변수의값
ex) Maximize p = 6x + 14y + 13z
    Constraints: 0.5x + 2y + z ≤ 24
                 x + 2y + 4z ≤ 60
    n = 2, m = 3, a = [[0.5, 2, 1], [1, 2, 4]], b = [24, 60], c = [6, 14, 13]
*/

namespace simplex {
  using T = long double;
  const int N = 410, M = 30010;
  const T eps = 1e-7;
  int n, m;
  int Left[M], Down[N];
  T a[M][N], b[M], c[N], v, sol[N];

  bool eq(T a, T b) { return fabs(a - b) < eps;  }
  bool ls(T a, T b) { return a < b && !eq(a, b); }
```

```
void init(int p, int q) {
  n = p; m = q; v = 0;
  for(int i = 1; i <= m; i++){
    for(int j = 1; j <= n; j++) a[i][j]=0;
  }
  for(int i = 1; i <= m; i++) b[i]=0;
  for(int i = 1; i <= n; i++) c[i]=sol[i]=0;
}

void pivot(int x,int y) {
  swap(Left[x], Down[y]);
  T k = a[x][y]; a[x][y] = 1;
  vector<int> nz;
  for(int i = 1; i <= n; i++){
    a[x][i] /= k;
    if(!eq(a[x][i], 0)) nz.push_back(i);
  }
  b[x] /= k;

  for(int i = 1; i <= m; i++){
    if(i == x || eq(a[i][y], 0)) continue;
    k = a[i][y]; a[i][y] = 0;
    b[i] -= k*b[x];
    for(int j : nz) a[i][j] -= k*a[x][j];
  }
  if(eq(c[y], 0)) return;
  k = c[y]; c[y] = 0;
  v += k*b[x];
  for(int i : nz) c[i] -= k*a[x][i];
}

// 0: found solution, 1: no feasible solution, 2: unbounded
int solve() {
  for(int i = 1; i <= n; i++) Down[i] = i;
  for(int i = 1; i <= m; i++) Left[i] = n+i;
  while(1) { // Eliminating negative b[i]
    int x = 0, y = 0;
    for(int i = 1; i <= m; i++) if (ls(b[i], 0) && (x == 0 || b[i] < b[x])) x
      = i;
    if(x == 0) break;
    for(int i = 1; i <= n; i++) if (ls(a[x][i], 0) && (y == 0 || a[x][i] < a[x
      ][y])) y = i;
    if(y == 0) return 1;
    pivot(x, y);
  }
  while(1) {
    int x = 0, y = 0;
    for(int i = 1; i <= n; i++)
      if (ls(0, c[i]) && (!y || c[i] > c[y])) y = i;
    if(y == 0) break;
    for(int i = 1; i <= m; i++)
      if (ls(0, a[i][y]) && (!x || b[i]/a[i][y] < b[x]/a[x][y])) x = i;
    if(x == 0) return 2;
    pivot(x, y);
  }
  for(int i = 1; i <= m; i++) if(Left[i] <= n) sol[Left[i]] = b[i];
```

```
    return 0;
  }
}
```

6.4 Gaussian Elimination

```
#define MAX_N 300       // adjust this value as needed
struct AugmentedMatrix { double mat[MAX_N][MAX_N + MAX_N + 10]; };
struct ColumnVector { double vec[MAX_N]; };

// 0 indexed row and column
AugmentedMatrix GaussianElimination(int N, AugmentedMatrix Aug) {
    // input: N X 2N matrix  [A I], output: [I invA]

    // forward eliminataion phase
    for(int i=0;i<N;i++){
        int l = i;
        // which row has largest column value
        for(int j=i+1;j<N;j++)
            if( fabs(Aug.mat[j][i]) > fabs(Aug.mat[l][i]) )
                l = j;
        // swap this pivot row to minimize error
        for(int k=i;k<2*N;k++)
            swap(Aug.mat[i][k],Aug.mat[l][k]);
        // calculate forward elimination
        for(int j=i+1;j<N;j++)
            for(int k=2*N-1;k>=i;k--)
                Aug.mat[j][k] -= Aug.mat[i][k] * Aug.mat[j][i] / Aug.mat[i][i];
    }

    // normalize pivots
    for(int i=0;i<N;i++)
        for(int j=2*N-1;j>=i;j--)
            Aug.mat[i][j] /= Aug.mat[i][i];

    // backward elimination
    for(int i=N-1;i>0;i--)
        for(int j=i-1;j>=0;j--)
            for(int k=2*N-1;k>=i;k--)
                Aug.mat[j][k] -= Aug.mat[i][k] * Aug.mat[j][i] / Aug.mat[i][i];


    return Aug;
}

int main() {

    AugmentedMatrix Aug;
    int N; geti(N);
    rep(i,N) rep(j,N) scanf("%lf",&Aug.mat[i][j]);
    for(int i=N;i<2*N;i++) Aug.mat[i-N][i] = 1;

    AugmentedMatrix res = GaussianElimination(N, Aug);

    // Print inversion of A
    for(int i=0;i<N;i++){
        for(int j=N;j<2*N;j++) printf("%f ",res.mat[i][j]);
        printf("\n");
    }


    return 0;
}
```

6.5 Prime Algorithms

```
typedef long long ll;
using namespace std;

ll gcd(ll a, ll b) {
    if (b == 0)
        return a;
    return gcd(b, a%b);
}

namespace miller_rabin {
    ll mul(ll x, ll y, ll mod) { return (__int128)x * y % mod; }
    //ll mul(ll x, ll y, ll mod) { return x * y % mod; }
    ll ipow(ll x, ll y, ll p) {
        ll ret = 1, piv = x % p;
        while (y) {
            if (y & 1) ret = mul(ret, piv, p);
            piv = mul(piv, piv, p);
            y >>= 1;
        }
        return ret;
    }
    bool miller_rabin(ll x, ll a) {
        if (x % a == 0) return 0;
        ll d = x - 1;
        while (1) {
            ll tmp = ipow(a, d, x);
            if (d & 1) return (tmp != 1 && tmp != x - 1);
            else if (tmp == x - 1) return 0;
            d >>= 1;
        }
    }
    bool isprime(ll x) {
        for (auto &i : { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 }) {
            if (x == i) return 1;
            if (x > 40 && miller_rabin(x, i)) return 0;
        }
        if (x <= 40) return 0;
        return 1;
    }
}
namespace pollard_rho {
    ll f(ll x, ll n, ll c) {
        return (c + miller_rabin::mul(x, x, n)) % n;
    }
    void rec(ll n, vector<ll> &v) {
        if (n == 1) return;
        if (n % 2 == 0) {
            v.push_back(2);
            rec(n / 2, v);
            return;
        }
        if (miller_rabin::isprime(n)) {
            v.push_back(n);
            return;
        }
        ll a, b, c;
        while (1) {
            a = rand() % (n - 2) + 2;
            b = a;
            c = rand() % 20 + 1;
            do {
                a = f(a, n, c);
                b = f(f(b, n, c), n, c);
            } while (gcd(abs(a - b), n) == 1);
            if (a != b) break;
        }
        ll x = gcd(abs(a - b), n);
        rec(x, v);
        rec(n / x, v);
    }
    vector<ll> factorize(ll n) {
        vector<ll> ret;
        rec(n, ret);
        sort(ret.begin(), ret.end());
        return ret;
    }
};

int main() {
    vector<ll> res;
    ll num;
    scanf("%lld", &num);
    res = pollard_rho::factorize(num);
    for (int i = 0; i < res.size(); ++i)
        printf("%lld\n", res[i]);
}
```

7 Miscelleneous

7.1 Hungarian

```
/*
Tests
http://www.spoj.com/problems/GREED/
https://www.acmicpc.net/problem/8992
SRM 506 mid

Time complexity O(n^3)

Usage
MinWeightBipartiteMatch matcher(n);
for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) matcher.weights[i][j] =
  SOMETHING;
cost_t total = matcher.solve();

See matcher.match(row -> col) and matcher.matched(col -> row) for actual match
*/

struct MinWeightBipartiteMatch
{
  typedef long long cost_t;

  cost_t max_cost() const { return numeric_limits<cost_t>::max(); }

  // input
  int n;
  vector<vector<cost_t>> weights;
  // output
  vector<int> match, matched;

  MinWeightBipartiteMatch(int n) :
    n(n), match(n), matched(n), weights(n, vector<cost_t>(n))
  {

  }

  void resize(int n) {
    this->n = n;
    match.resize(n);
    matched.resize(n);
    weights.resize(n);
    for (int i = 0; i < n; i++) {
      weights[i].resize(n);
    }
  }

  /* for solve() */
  vector<cost_t> slack;
  vector<cost_t> potential_row, potential_col;
  vector<int> reach_row, reach_col;
  int rcnt;
  vector<int> from;
  void found_match(int r, int c) {
    do {
      int old_match = match[r];
      match[r] = c;
      matched[c] = r;
      tie(r, c) = make_pair(from[r], old_match);
    } while (r >= 0 && c >= 0);
  }

  void augment(int row_to_match) {
    slack.resize(n);
    for (int c = 0; c < n; c++) {
      slack[c] = weights[row_to_match][c] - potential_row[row_to_match] -
        potential_col[c];
    }
    ++rcnt;
    vector<int> q; q.reserve(n);
    int h = 0;
    q.push_back(row_to_match);
    reach_row[row_to_match] = rcnt;
    from[row_to_match] = -1;
    for (;;) {
      while (h < q.size()) {
        int r = q[h++];
        for (int c = 0; c < n; c++) {
          cost_t gap = weights[r][c] - potential_row[r] - potential_col[c];
          slack[c] = min(slack[c], gap);
          if (gap != cost_t()) continue;
          int next = matched[c];
          if (next < 0) {
            found_match(r, c);
            return;
          }
          reach_col[c] = rcnt;
          if (reach_row[next] == rcnt) continue;
          q.push_back(next);
          reach_row[next] = rcnt;
          from[next] = r;
        }
      }
      cost_t delta = max_cost();
      for (int c = 0; c < n; c++) {
        if (reach_col[c] == rcnt) continue; // non-covered -> continue
        delta = min(delta, slack[c]);
      }
      for (int r = 0; r < n; r++) {
        if (reach_row[r] == rcnt) continue;
        potential_row[r] -= delta;
      }
      for (int c = 0; c < n; c++) {
        if (reach_col[c] == rcnt) continue;
        potential_col[c] += delta;
        slack[c] -= delta;
      }
      int lastsize = q.size();
      for (int c = 0; c < n; c++) {
        if (reach_col[c] == rcnt) continue;
        if (slack[c] != cost_t()) continue;
        int next = matched[c];
        if (next >= 0 && reach_row[next] == rcnt) continue;
        for (int qi = 0; qi < lastsize; qi++) {
          int r = q[qi];
          cost_t gap = weights[r][c] - potential_row[r] - potential_col[c];
          if (gap != cost_t()) continue;
          if (next < 0) {
            found_match(r, c);
            return;
          }
          reach_col[c] = rcnt;
          q.push_back(next);
          reach_row[next] = rcnt;
          from[next] = r;
          break;
        }
      }
    }
  }

  void initialize() {
    potential_row.assign(n, cost_t());
    potential_col.assign(n, cost_t());
    match.assign(n, -1);
    matched.assign(n, -1);
    reach_row.assign(n, 0);
    reach_col.assign(n, 0);
    from.resize(n);
    rcnt = 1;
    for (int i = 0; i < n; i++) {
      cost_t row_min_weight = *min_element(weights[i].begin(), weights[i].end())
        ;
      potential_row[i] = row_min_weight;
    }
    for (int i = 0; i < n; i++) {
      cost_t col_min_weight = weights[0][i] - potential_row[0];
      for (int j = 1; j < n; j++) col_min_weight = min(col_min_weight, weights[j
        ][i] - potential_row[j]);
      potential_col[i] = col_min_weight;
    }
  }

  cost_t solve() {
    initialize();
    for (int row_to_match = 0; row_to_match < n; row_to_match++) {
      augment(row_to_match);
    }
    cost_t ans = cost_t();
    for (auto v : potential_row) ans += v;
    for (auto v : potential_col) ans += v;
    return ans;
  }
};
```

7.2 LiChao Tree

```
// LiChaoTree for dynamic CHT trick
// This example maintains CHT for finding MAXIMUM of corresponding x
// op=1 : add ax + b into CHT
// op=2 : find max value of position x
// https://cp-algorithms.com/geometry/convex_hull_trick.html
ll f(Pll line, ll x){
    return line.Fi*x + line.Se;
}
```

```
vector<ll> xlist;
struct LiChaoTree{
    int n; vector<Pll> d;
    void init(int x){
        n = 1; while (n < x) n *= 2;
        d.resize(n*2+10);
        for(auto& e : d){
            e = {0, -3*(1e18)};
        }
    }

    void insert(int node, int nL, int nR, Pll newline){
        if( nL == nR ){
            if( f(d[node], xlist[nL]) < f(newline, xlist[nL]) ) d[node] =
              newline;
            return;
        }
        bool left = f(d[node], xlist[nL]) < f(newline, xlist[nL]);
        bool right = f(d[node], xlist[nR]) < f(newline, xlist[nR]);

        // take upper, lower line based on leftmost point of the segment
        Pll upper = d[node], lower = newline;
        if( left ) swap(upper, lower);

        // one line totally cover another line
        if( left == right ){
            d[node] = upper; return;
        }

        int m = (nL+nR)/2;
        // intersection in left half segment
        if( f(upper, xlist[m]) <= f(lower, xlist[m]) ){
            d[node] = lower;
            insert(node*2,nL,m, upper);
        }
        // intersection in right half segment
        else{
            d[node] = upper;
            insert(node*2+1,m+1,nR,lower);
        }
    }

    ll query(int node, int nL, int nR, int pos){
        if( nL == nR ) return f(d[node], xlist[pos]);

        int m = (nL+nR)/2;
        ll nval = -3*(1e18);
        if( pos <= m ) nval = query(node*2, nL, m, pos);
        else nval = query(node*2+1, m+1, nR, pos);

        return max(nval, f(d[node], xlist[pos]) );
    }

};

int main(){
    int Q; scanf("%d",&Q);
    vector<pair<int,Pll>> qlist;
    repp(q,Q){
        int op; scanf("%d",&op);
        if( op == 1 ){
            ll a,b; scanf("%lld%lld",&a,&b);
            qlist.push_back({1,{a,b}});
        }
        else{
            ll x; scanf("%lld",&x);
            xlist.push_back(x);
            qlist.push_back({2,{x,x}});
        }
    }

    xlist.push_back(-2*(1e12) - 10);
    sort(all(xlist));
    xlist.erase(unique(all(xlist)), xlist.end());
    LiChaoTree tree;
    tree.init( sz(xlist)+1 );

    // careful to put padding into xlist
    // so that it fits to tree size
    while( sz(xlist) < tree.n+5 )  xlist.push_back(2*(1e12));

    for(auto q : qlist){
        if( q.Fi == 1 ){
            tree.insert(1,1,tree.n,q.Se);
        }
        if( q.Fi == 2 ){
            int pos = lower_bound(all(xlist), q.Se.Fi) - xlist.begin();
            printf("%lld\n",tree.query(1,1,tree.n,pos));
        }
    }

}
```

7.3 Persistence Segment Tree

```
int n, cnt;
int root[MAXN];

struct node {
    int sum, left, right;
} tree[3 * MAXN * LOGN];

int build(int l = 0, int r = n) {
    int idx = ++cnt;
    if(r - l <= 1) {
        tree[idx] = {0, 0, 0};
        return idx;
    }
    int mid = (l + r) >> 1;
    tree[idx] = {0, build(l, mid), build(mid, r)};
    return idx;
}

int update(int x, int prev, int l = 0, int r = n) {
    if(x < l || r <= x) return prev;
    int idx = ++cnt;
    if(r - l <= 1) {
        tree[idx] = {1, 0, 0};
        return idx;
    }

    int mid = (l + r) >> 1;
    int L = update(x, tree[prev].left, l, mid);
    int R = update(x, tree[prev].right, mid, r);
    tree[idx] = {tree[L].sum + tree[R].sum, L, R};
    return idx;
}

int query(int x, int y, int k, int l = 0, int r = n) {
    if(r - l <= 1) return l;
    int mid = (l + r) >> 1;
    int leftSum = tree[tree[y].left].sum - tree[tree[x].left].sum;
    if(leftSum >= k)
        return query(tree[x].left, tree[y].left, k, l, mid);
    else
        return query(tree[x].right, tree[y].right, k - leftSum, mid, r);
}



int a[MAXN], rev[MAXN];
map<int, int> M;

int main() {
    int q;
    geti(n, q);
    for(int i = 1; i <= n; i++) {
        geti(a[i]);
        rev[i-1] = a[i];
    }
    sort(rev, rev + n);
    for(int i = 0; i < n; i++)
        M[rev[i]] = i;
    for(int i = 1; i <= n; i++)
        a[i] = M[a[i]];

    root[0] = build();
    for(int i = 1; i <= n; i++)
        root[i] = update(a[i], root[i-1]);

    while(q--) {
        int i, j, k;
        geti(i, j, k);
        printf("%d\n", rev[query(root[i-1], root[j], k)]);
    }
}
```

7.4 XOR FFT

```
#include <cstdio>
#include <complex>

const int SZ = 20, N = 1 << SZ;

using namespace std;

int Rev(int x) {
    int i, r = 0;
    for (i = 0; i < SZ; i++) {
        r = r << 1 | x & 1;
        x >>= 1;
    }
    return r;
}

void FFT(int *a, bool f) {
    int i, j, k, z;
    for (i = 0; i < N; i++) {
        j = Rev(i);
        if (i < j) {
            z = a[i];
            a[i] = a[j];
            a[j] = z;
        }
    }
    for (i = 1; i < N; i <<= 1) for (j = 0; j < N; j += i << 1) for (k = 0; k <
      i; k++) {
        z = a[i + j + k];
        a[i + j + k] = a[j + k] - z;
        a[j + k] += z;
    }
    if (f) for (i = 0; i < N; i++) a[i] /= N;
}

int X[N];

int main() {
    int i, n;
    scanf("%d", &n);
    for (i = 0; i < 1 << n; i++) scanf("%d", &X[i]);
    FFT(X, false);
    for (i = 0; i < N; i++) X[i] *= X[i];
    FFT(X, true);
    for (i = 0; i < 1 << n; i++) printf("%d ", X[i]);
}
```

7.5 NTT

```
#include <cstdio>

const int A = 7, B = 26, P = A << B | 1, R = 3;
const int SZ = 20, N = 1 << SZ;
```

```
int Pow(int x, int y) {
    int r = 1;
    while (y) {
        if (y & 1) r = (long long)r * x % P;
        x = (long long)x * x % P;
        y >>= 1;
    }
    return r;
}

void FFT(int *a, bool f) {
    int i, j, k, x, y, z;
    j = 0;
    for (i = 1; i < N; i++) {
        for (k = N >> 1; j >= k; k >>= 1) j -= k;
        j += k;
        if (i < j) {
            k = a[i];
            a[i] = a[j];
            a[j] = k;
        }
    }
    for (i = 1; i < N; i <<= 1) {
        x = Pow(f ? Pow(R, P - 2) : R, P / i >> 1);
        for (j = 0; j < N; j += i << 1) {
            y = 1;
            for (k = 0; k < i; k++) {
                z = (long long)a[i | j | k] * y % P;
                a[i | j | k] = a[j | k] - z;
                if (a[i | j | k] < 0) a[i | j | k] += P;
                a[j | k] += z;
                if (a[j | k] >= P) a[j | k] -= P;
                y = (long long)y * x % P;
            }
        }
    }
    if (f) {
        j = Pow(N, P - 2);
        for (i = 0; i < N; i++) a[i] = (long long)a[i] * j % P;
    }
}

int X[N];

int main() {
    int i, n;
    scanf("%d", &n);
    for (i = 0; i <= n; i++) scanf("%d", &X[i]);
    FFT(X, false);
    for (i = 0; i < N; i++) X[i] = (long long)X[i] * X[i] % P;
    FFT(X, true);
    for (i = 0; i <= n + n; i++) printf("%d ", X[i]);
}
```

7.6 2D FFT

```
const double EPS = 0.00001;

typedef complex<double> base;

void fft(vector<base> &a, bool invert){
    int n = a.size();
    for(int i=1,j=0;i<n;i++){
        int bit = n >> 1;
        for (;j>=bit;bit>>=1)j -= bit;
            j += bit;
        if (i < j) swap(a[i], a[j]);
    }
    for(int len=2;len<=n;len<<=1){
        double ang = 2*acos(-1)/len*(invert?-1:1);
        base wlen(cos(ang),sin(ang));
        for(int i=0;i<n;i+=len){
            base w(1);
            for(int j=0;j<len/2;j++){
                base u = a[i+j], v = a[i+j+len/2]*w;
                a[i+j] = u+v;
                a[i+j+len/2] = u-v;
                w *= wlen;
            }
        }
    }
    if (invert) {
        for(int i=0;i<n;i++) a[i] /= n;
    }
}

void multiply(const vector<int> &a, const vector<int> &b, vector<int> &res){
    vector<base> fa(a.begin(), a.end()), fb(b.begin(),b.end());
    int n = 1;
    while(n < max(a.size(), b.size())) n <<= 1;
    n <<= 1;
    fa.resize(n); fb.resize(n);
    fft(fa,false);fft(fb,false);
    for(int i=0;i<n;i++) fa[i] *= fb[i];
        fft(fa,true);
    res.resize(n);
    for(int i=0;i<n;i++) res[i] = int(fa[i].real() + (fa[i].real() > 0 ? 0.5 :
      -0.5));
}

void multiply_complex(const vector<base> &a, const vector<base> &b, vector<base>
   &res){
    vector<base> fa(a.begin(), a.end()), fb(b.begin(),b.end());
    int n = 1;
    while(n < max(a.size(), b.size())) n <<= 1;
    n <<= 1;
    fa.resize(n); fb.resize(n);
    fft(fa,false);fft(fb,false);
    for(int i=0;i<n;i++) fa[i] *= fb[i];
        fft(fa,true);
    res.resize(n);
    for(int i=0;i<n;i++) res[i] = fa[i];
}

const int MAXN = 405;
const int LOGN = 19;

string S[MAXN], T[MAXN];

int main() {
    int n, m;
    geti(n, m);
    for(int i = 0; i < n; i++)
        cin >> S[i];
    int r, c;
    geti(r, c);
    for(int i = 0; i < r; i++)
        cin >> T[i];


    int p = 1, q = 1;
    while(q < m+c) q <<= 1;
    while(p < n+r) p <<= 1;

    vector<vector<base>> a(p, vector<base>(q)), b(p, vector<base>(q));
    for(int i = 0; i < p; i++) {
        for(int j = 0; j < q; j++) {
            int t = S[i%n][j%m] - 'a';
            double ang = 2*acos(-1)*t/26;
            a[i][j] = base(cos(ang), sin(ang));
        }
    }
    int cnt = 0;
    for(int i = 0; i < r; i++) {
        for(int j = 0; j < c; j++) {
            if(T[i][j] != '?') {
                cnt++;
                int t = T[i][j] - 'a';
                double ang = 2*acos(-1)*t/26;
                b[(r-1)-i][(c-1)-j] = base(cos(-ang), sin(-ang));
            }
        }
    }

    vector<vector<base>> fa, fb, res;
    for(int i = 0; i < p; i++) {
        vector<base> ta(a[i].begin(), a[i].end()), tb(b[i].begin(), b[i].end());
        fft(ta, false);
        fft(tb, false);
        fa.push_back(ta);
        fb.push_back(tb);
    }

    for(int j = 0; j < q; j++) {
        vector<base> ta(p), tb(p), tmp;
        for(int i = 0; i < p; i++) {
            ta[i] = fa[i][j];
            tb[i] = fb[i][j];
        }
        multiply_complex(ta, tb, tmp);
        if(j == 0)
            res.resize(tmp.size(), vector<base>(q));

        for(int i = 0; i < res.size(); i++)
            res[i][j] = tmp[i];
    }

    for(int i = 0; i < res.size(); i++)
        fft(res[i], true);

    for(int i = 0; i < n; i++) {
        for(int j = 0; j < m; j++) {
            if(abs(res[i+r-1][j+c-1].real() - cnt) < EPS && abs(res[i+r-1][j+c
              -1].imag()) < EPS) printf("1");
            else printf("0");
        }
        printf("\n");
    }
}
```

7.7 Order Statistic Tree

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

// Need this
// We can run this code on codeforces
// http://codeforces.com/blog/entry/11080
using namespace __gnu_pbds;

typedef tree<
int,
null_type,
less<int>,
rb_tree_tag,
tree_order_statistics_node_update>
ordered_set;

int main(){
    ordered_set X;
    X.insert(1);
    X.insert(2);
    X.insert(4);
    X.insert(8);
    X.insert(16);

    cout<<*X.find_by_order(1)<<endl; // 2
    cout<<*X.find_by_order(2)<<endl; // 4
    cout<<*X.find_by_order(4)<<endl; // 16
    cout<<(end(X)==X.find_by_order(6))<<endl; // true
    cout<<X.order_of_key(-5)<<endl;  // 0
    cout<<X.order_of_key(1)<<endl;   // 0
    cout<<X.order_of_key(3)<<endl;   // 2
    cout<<X.order_of_key(4)<<endl;   // 2
    cout<<X.order_of_key(400)<<endl; // 5

}
```

7.8 BITSET

```
#define M 32
int main()
{
    // default constructor initializes with all bits 0
    bitset<M> bset1;

    // bset2 is initialized with bits of 20
    bitset<M> bset2(20);

    // bset3 is initialized with bits of specified binary string
    bitset<M> bset3(string("1100"));

    // cout prints exact bits representation of bitset
    cout << bset1 << endl;  // 00000000000000000000000000000000
    cout << bset2 << endl;  // 00000000000000000000000000010100
    cout << bset3 << endl;  // 00000000000000000000000000001100
    cout << endl;

    // declaring set8 with capacity of 8 bits

    bitset<8> set8;    // 00000000

    // setting first bit (or 6th index)
    set8[1] = 1;    // 00000010
    set8[4] = set8[1];   //  00010010
    cout << set8 << endl;

    // count function returns number of set bits in bitset
    int numberof1 = set8.count();

    // size function returns total number of bits in bitset
    // so there difference will give us number of unset(0)
    // bits in bitset
    int numberof0 = set8.size() - numberof1;
    cout << set8 << " has " << numberof1 << " ones and "
         << numberof0 << " zeros\n";

    // test function return 1 if bit is set else returns 0
    cout << "bool representation of " << set8 << " : ";
    for (int i = 0; i < set8.size(); i++)
        cout << set8.test(i) << " ";

    cout << endl;

    // any function returns true, if atleast 1 bit
    // is set
    if (!set8.any())
        cout << "set8 has no bit set.\n";

    if (!bset1.any())
        cout << "bset1 has no bit set.\n";

    // none function returns true, if none of the bit
    // is set
    if (!bset1.none())
        cout << "bset1 has all bit set\n";

    // bset.set() sets all bits
    cout << set8.set() << endl;

    //  bset.set(pos, b) makes bset[pos] = b
    cout << set8.set(4, 0) << endl;

    // bset.set(pos) makes bset[pos] = 1  i.e. default
    // is 1
    cout << set8.set(4) << endl;

    // reset function makes all bits 0
    cout << set8.reset(2) << endl;
    cout << set8.reset() << endl;

    // flip function flips all bits i.e.  1 <-> 0
    // and  0 <-> 1
    cout << set8.flip(2) << endl;
    cout << set8.flip() << endl;

    // Converting decimal number to binary by using bitset
    int num = 100;
    cout  << "\nDecimal number: " << num
         << "  Binary equivalent: " << bitset<8>(num);

    return 0;
}

int main()
{
    bitset<4> bset1(9);      // bset1 contains 1001
    bitset<4> bset2(3);      // bset2 contains 0011

    // comparison operator
    cout << (bset1 == bset2) << endl;  // false 0
    cout << (bset1 != bset2) << endl;  // true  1

    // bitwise operation and assignment
    cout << (bset1 ^= bset2) << endl;  // 1010
    cout << (bset1 &= bset2) << endl;  // 0010
    cout << (bset1 |= bset2) << endl;  // 0011

    // left and right shifting
    cout << (bset1 <<= 2) << endl;     // 1100
    cout << (bset1 >>= 1) << endl;     // 0110

    // not operator
    cout << (~bset2) << endl;          // 1100

    // bitwise operator
    cout << (bset1 & bset2) << endl;    // 0010
    cout << (bset1 | bset2) << endl;    // 0111
    cout << (bset1 ^ bset2) << endl;    // 0101
}
```