Team notebook

Lazy Three

January 10, 2024

Contents

1 data-structures

1.1 DSU

```
struct DSU {
    int n;
    vector<int> parent;
    vector<int> size;
```

```
    DSU(int _n) : n(_n), parent(n), size(n, 1) {
        iota(parent.begin(), parent.end(), 0); }

    int find_set(int x) {
        if (parent[x] == x) return x;
        return parent[x] = find_set(parent[x]);
    }

    int getSize(int x) { return size[find_set(x)]; } // returns
        size of component of x

    void union_sets(int x, int y) {
        x = find_set(x);
        y = find_set(y);
        if (x == y) return;
        if (size[x] > size[y]) {
            parent[y] = x;
            size[x] += size[y];
        } else {
            parent[x] = y;
            size[y] += size[x];
        }
    }
};
```

1.2 Fenwick2D

```
const int mxn = 1000;
int grid[mxn + 1][mxn + 1];
int bit[mxn + 1][mxn + 1];
void update(int row, int col, int d){
    grid[row][col] += d;
    for(int i = row; i <= mxn; i += (i & -i))
        for(int j = col; j <= mxn; j += (j & -j))
            bit[i][j] += d;
}
int sum(int row, int col){
    //calculates sum from [1,1] till [row,col]
    int res = 0;
    for(int i = row; i > 0; i -= (i & -i))
        for(int j = col; j > 0; j -= (j & -j))
            res += bit[i][j];
    return res;
}
```

1.3 Mos

```
int BLOCK = DO_NOT_FORGET_TO_CHANGE_THIS;
struct Query{
    int l, r, id;
    Query(int _l, int _r, int _id) : l(_l), r(_r), id(_id) {}
    bool operator<(Query &o){
        int mblock = l / BLOCK, oblock = o.l / BLOCK;
        return (mblock < oblock) or
               (mblock == oblock and mblock % 2 == 0 and r < o.r)
                  or
               (mblock == oblock and mblock % 2 == 1 and r > o.r);
    };
};

// Solve
void solve(){

    vector<Query> queries; queries.reserve(q);
    for(int i=0; i < q; i++){
        int l, r; cin >> l >> r; l--, r--;
        queries.emplace_back(l, r, i);
    }
```

```
    sort(all(queries));

    int ans = 0;
    auto add = [&](int v){};
    auto rem = [&](int v){};

    vector<int> out(q); // Change out type if necessary
    int cur_l = 0, cur_r = -1;
    for(auto &[l, r, id] : queries){
        while(cur_l > l) add(--cur_l);
        while(cur_l < l) rem(cur_l++);
        while(cur_r < r) add(++cur_r);
        while(cur_r > r) rem(cur_r--);
        out[id] = ans;
    }
}
```

1.4 RMQ

```
template<class T>
struct RMQ {
        vector<vector<T>> jmp;
        RMQ(const vector<T>& V) : jmp(1, V) {
                for (int pw = 1, k = 1; pw * 2 <= sz(V); pw *= 2,
                    ++k) {
                        jmp.emplace_back(sz(V) - pw * 2 + 1);
                        rep(j,0,sz(jmp[k]))
                                jmp[k][j] = min(jmp[k - 1][j],
                                    jmp[k - 1][j + pw]);
                }
        }
        T query(int a, int b) {
        b++;
                assert(a < b); // or return inf if a == b
                int dep = 31 - __builtin_clz(b - a);
                return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
        }
};
```

1.5 SegTree

```
template <typename T, typename F>
struct SegTree {
    int n;
    vector<T> t;
    const T id;
    F f;
    SegTree(const vector<T> &a, T id, F f) : n(sz(a)), t(2 * n),
        id(id), f(f) {
        for (int i = 0; i < n; i++) t[n + i] = a[i];
        for (int i = n - 1; i >= 1; i--)
            t[i] = f(t[2 * i], t[2 * i + 1]);
    }

    T query(int l, int r) {
        T resl(id), resr(id);
        for (l += n, r += n; l <= r; l >>= 1, r >>= 1) {
            if (l == r) {
                resl = f(resl, t[l]);
                break;
            }
            if (l & 1) resl = f(resl, t[l++]);
            if (!(r & 1)) resr = f(t[r--], resr);
        }
        return f(resl, resr);
    }

    void update(int v, T value) {
```

```
        for (t[v += n] = value; v >>= 1;)
            t[v] = f(t[2 * v], t[2 * v + 1]);
    }
};
```

1.6 fenwick

```
template <typename T>
struct Fenwick {
    vector<T> bit;
    vector<T>& original;
    Fenwick(vector<T>& _arr) : bit(_arr.size(), 0LL),
        original(_arr) {
        int n = sz(_arr);
        for (int i = 0; i < n; i++) {
            bit[i] = bit[i] + _arr[i];
            if ((i | (i + 1)) < n) bit[(i | (i + 1))] = bit[(i |
                (i + 1))] + bit[i];
        }
    }

    // returns smallest index i, st. sum[0..i] >= x, returns -1
        if no such i exists
    // returns n if x >= sum of array
    // ASSUMES NON NEGATIVE ENTRIES IN TREE
    int lower_bound(int x) {
        if (x < 0) return -1;
        if (x == 0) return 0;
        int pos = 0;
        for (int pw = 1LL << 20; pw; pw >>= 1)
            if (pw + pos <= sz(bit) and bit[pos + pw - 1] < x)
                pos += pw, x -= bit[pos - 1];
        return pos;
    }

    T query(int r) {
        assert(r < sz(bit));
        int ret = 0;
        for (r++; r > 0; r &= r - 1) ret += bit[r - 1];
        return ret;
    }

    T query(int l, int r) {
        T ret = query(r);
        if (l != 0) ret -= query(l - 1);
        return ret;
    }

    void update(int i, int x) {
        int n = bit.size();
        T diff = x - original[i];
        original[i] = x;
        for (; i < n; i = i | i + 1) bit[i] += diff;
    }
};
```

1.7 lazy

```
template<typename T, typename U> struct seg_tree_lazy {
    int S, H;

    T zero;
    vector<T> value;

    U noop;
    vector<bool> dirty;
    vector<U> prop;
```

```
seg_tree_lazy(int _S, T _zero = T(), U _noop = U()) {
    zero = _zero, noop = _noop;
    for (S = 1, H = 1; S < _S; ) S *= 2, H++;

    value.resize(2*S, zero);
    dirty.resize(2*S, false);
    prop.resize(2*S, noop);
}

void set_leaves(vector<T> &leaves) {
    copy(leaves.begin(), leaves.end(), value.begin() + S);
    for (int i = S - 1; i > 0; i--)
        value[i] = value[2 * i] + value[2 * i + 1];
}

void apply(int i, U &update) {
    value[i] = update(value[i]);
    if(i < S) {
        prop[i] = prop[i] + update;
        dirty[i] = true;
    }
}

void rebuild(int i) {
    for (int l = i/2; l; l /= 2) {
        T combined = value[2*l] + value[2*l+1];
        value[l] = prop[l](combined);
    }
}

void propagate(int i) {
    for (int h = H; h > 0; h--) {
        int l = i >> h;

        if (dirty[l]) {
            apply(2*l, prop[l]);
            apply(2*l+1, prop[l]);

            prop[l] = noop;
            dirty[l] = false;
        }
    }
}

void upd(int i, int j, U update) {
    i += S, j += S;
    propagate(i), propagate(j);

    for (int l = i, r = j; l <= r; l /= 2, r /= 2) {
        if((l&1) == 1) apply(l++, update);
        if((r&1) == 0) apply(r--, update);
    }

    rebuild(i), rebuild(j);
}

T query(int i, int j){
    i += S, j += S;
    propagate(i), propagate(j);

    T res_left = zero, res_right = zero;
    for(; i <= j; i /= 2, j /= 2){
        if((i&1) == 1) res_left = res_left + value[i++];
        if((j&1) == 0) res_right = value[j--] + res_right;
    }
    return res_left + res_right;
}
};

struct node {
```

```
    int sum, width;

    node operator+(const node &n) {
        // Change 1
        return { sum + n.sum, width + n.width };
    }
};

struct update {
    bool type; // 0 for add, 1 for reset
    int value;

    node operator()(const node &n) { // apply update on n
        // Change 2
        if (type) return { n.width * value, n.width };
        else return { n.sum + n.width * value, n.width };
    }

    update operator+(const update &u) { // u is the recent
        update, *this is the older update
        // Change 3
        if (u.type) return u;
        return { type, value + u.value };
    }
};
```

2 flags

```
1. slow_run: Enables a ton of checks, can catch silly mistakes
    (off by 1, overflow, non void function that didnt return at
    runtime etc)
g++ -Wall -Wextra -pedantic -std=c++17 -O2 -Wshadow -Wformat=2
    -Wfloat-equal -Wconversion -Wlogical-op -Wshift-overflow=2
    -Wduplicated-cond -Wcast-qual -Wcast-align -D_GLIBCXX_DEBUG
    -D_GLIBCXX_DEBUG_PEDANTIC -D_FORTIFY_SOURCE=2
    -fsanitize=address -fsanitize=undefined
    -fno-sanitize-recover -fstack-protector -DLOCAL A.cpp -o A

2. fast_run: for fast runtime
g++ -std=c++17 -O2 -DLOCAL A.cpp -o A

3. debug: has debug flags enabled on top of the checks
g++ -std=c++17 -Wall -Wextra -pedantic -Wshadow
    -Wno-unused-parameter -Wno-unused-variable -Wformat=2
    -Wfloat-equal -Wconversion -Wlogical-op -Wshift-overflow=2
    -Wduplicated-cond -Wcast-qual -Wcast-align -D_GLIBCXX_DEBUG
    -D_GLIBCXX_DEBUG_PEDANTIC -D_FORTIFY_SOURCE=2
    -fsanitize=address -fsanitize=undefined
    -fno-sanitize-recover -fstack-protector -DLOCAL
    -fdiagnostics-color=always A.cpp -o A

## Put this in end of ~/.bashrc
alias s="g++ -Wall -Wextra -pedantic -std=c++17 -O2 -Wshadow
    -Wformat=2 -Wfloat-equal -Wconversion -Wlogical-op
    -Wshift-overflow=2 -Wduplicated-cond -Wcast-qual
    -Wcast-align -D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC
    -D_FORTIFY_SOURCE=2 -fsanitize=address -fsanitize=undefined
    -fno-sanitize-recover -fstack-protector -DLOCAL A.cpp"
alias f="g++ -std=c++17 -O2 -DLOCAL A.cpp"
alias d="g++ -std=c++17 -Wall -Wextra -pedantic -Wshadow
    -Wno-unused-parameter -Wno-unused-variable -Wformat=2
    -Wfloat-equal -Wconversion -Wlogical-op -Wshift-overflow=2
    -Wduplicated-cond -Wcast-qual -Wcast-align -D_GLIBCXX_DEBUG
    -D_GLIBCXX_DEBUG_PEDANTIC -D_FORTIFY_SOURCE=2
    -fsanitize=address -fsanitize=undefined
    -fno-sanitize-recover -fstack-protector -DLOCAL
    -fdiagnostics-color=always"
```

3 geometry

3.1 Point

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
	typedef Point P;
	T x, y;
	explicit Point(T x=0, T y=0) : x(x), y(y) {}
	bool operator<(P p) const { return tie(x,y) <
	    tie(p.x,p.y); }
	bool operator==(P p) const { return
	    tie(x,y)==tie(p.x,p.y); }
	P operator+(P p) const { return P(x+p.x, y+p.y); }
	P operator-(P p) const { return P(x-p.x, y-p.y); }
	P operator*(T d) const { return P(x*d, y*d); }
	P operator/(T d) const { return P(x/d, y/d); }
	T dot(P p) const { return x*p.x + y*p.y; }
	T cross(P p) const { return x*p.y - y*p.x; }
	T cross(P a, P b) const { return
	    (a-*this).cross(b-*this); }
	T dist2() const { return x*x + y*y; }
	double dist() const { return sqrt((double)dist2()); }
	// angle to x-axis in interval [-pi, pi]
	double angle() const { return atan2(y, x); }
	P unit() const { return *this/dist(); } // makes dist()=1
	P perp() const { return P(-y, x); } // rotates +90 degrees
	P normal() const { return perp().unit(); }
	// returns point rotated 'a' radians ccw around the origin
	P rotate(double a) const {
		return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
	friend ostream& operator<<(ostream& os, P p) {
		return os << "(" << p.x << "," << p.y << ")"; }
};
```

3.2 closest-pair

```
// Requires point
typedef Point<int> P;
pair<P, P> closest(vector<P> v) {
	assert(sz(v) > 1);
	set<P> S;
	sort(all(v), [](P a, P b) { return a.y < b.y; });
	pair<int, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
	int j = 0;
	for (P p : v) {
		P d{1 + (int)sqrtl(ret.first), 0};
		while (v[j].y <= p.y - d.x) S.erase(v[j++]);
		auto lo = S.lower_bound(p - d), hi =
		    S.upper_bound(p + d);
		for (; lo != hi; ++lo)
			ret = min(ret, {(*lo - p).dist2(), {*lo,
			    p}});
		S.insert(p);
	}
	return ret.second;
}
```

3.3 convex-hull

```
// Needs point
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
	if (sz(pts) <= 1) return pts;
	sort(all(pts));
```

```
        vector<P> h(sz(pts)+1);
        int s = 0, t = 0;
        for (int it = 2; it--; s = --t, reverse(all(pts)))
                for (P p : pts) {
                        while (t >= s + 2 && h[t-2].cross(h[t-1],
                            p) <= 0) t--;
                        h[t++] = p;
                }
        return {h.begin(), h.begin() + t - (t == 2 && h[0] ==
            h[1])};
}
```

4 graph

4.1 CostScalingMCMF

```
template <const int MAX_N, typename flow_t, typename cost_t,
    flow_t FLOW_INF, cost_t COST_INF, const int SCALE = 16>
struct CostScalingMCMF {
#define sz(a) a.size()
#define zero_stl(v, sz) fill(v.begin(), v.begin() + (sz), 0)
  struct Edge {
    int v;
    flow_t c;
    cost_t d;
    int r;
    Edge() = default;
    Edge(int v, flow_t c, cost_t d, int r) : v(v), c(c), d(d),
        r(r) {}
  };
  vector<Edge> g[MAX_N];
  cost_t negativeSelfLoop;
  array<cost_t, MAX_N> pi, excess;
  array<int, MAX_N> level, ptr;
  CostScalingMCMF() { negativeSelfLoop = 0; }
  void clear() {
    negativeSelfLoop = 0;
    for (int i = 0; i < MAX_N; i++) g[i].clear();
  }
  void addEdge(int s, int e, flow_t cap, cost_t cost) {
    if (s == e) {
      if (cost < 0) negativeSelfLoop += cap * cost;
      return;
    }
    g[s].push_back(Edge(e, cap, cost, sz(g[e])));
    g[e].push_back(Edge(s, 0, -cost, sz(g[s]) - 1));
  }
  flow_t getMaxFlow(int V, int S, int T) {
    auto BFS = [&]() {
      zero_stl(level, V);
      queue<int> q;
      q.push(S);
      level[S] = 1;
      for (q.push(S); !q.empty(); q.pop()) {
        int v = q.front();
        for (const auto &e : g[v])
          if (!level[e.v] && e.c) q.push(e.v), level[e.v] =
              level[v] + 1;
      }
      return level[T];
    };
    function<flow_t(int, flow_t)> DFS = [&](int v, flow_t fl) {
      if (v == T || fl == 0) return fl;
      for (int &i = ptr[v]; i < (int)g[v].size(); i++) {
        Edge &e = g[v][i];
        if (level[e.v] != level[v] + 1 || !e.c) continue;
        flow_t delta = DFS(e.v, min(fl, e.c));
        if (delta) {
```

```
        e.c -= delta;
        g[e.v][e.r].c += delta;
        return delta;
      }
    }
    return flow_t(0);
  };
  flow_t maxFlow = 0, tmp = 0;
  while (BFS()) {
    zero_stl(ptr, V);
    while ((tmp = DFS(S, FLOW_INF))) maxFlow += tmp;
  }
  return maxFlow;
}
pair<flow_t, cost_t> maxflow(int N, int S, int T) {
  flow_t maxFlow = 0;
  cost_t eps = 0, minCost = 0;
  stack<int, vector<int>> stk;
  auto c_pi = [&](int v, const Edge &edge) { return edge.d +
      pi[v] - pi[edge.v]; };
  auto push = [&](int v, Edge &edge, flow_t delta, bool flag) {
    delta = min(delta, edge.c);
    edge.c -= delta;
    g[edge.v][edge.r].c += delta;
    excess[v] -= delta;
    excess[edge.v] += delta;
    if (flag && 0 < excess[edge.v] && excess[edge.v] <= delta)
        stk.push(edge.v);
  };
  auto relabel = [&](int v, cost_t delta) { pi[v] -= delta +
      eps; };
  auto lookAhead = [&](int v) {
    if (excess[v]) return false;
    cost_t delta = COST_INF;
    for (auto &e : g[v]) {
      if (e.c <= 0) continue;
      cost_t cp = c_pi(v, e);
      if (cp < 0)
        return false;
      else
        delta = min(delta, cp);
    }
    relabel(v, delta);
    return true;
  };
  auto discharge = [&](int v) {
    cost_t delta = COST_INF;
    for (int i = 0; i < sz(g[v]); i++) {
      Edge &e = g[v][i];
      if (e.c <= 0) continue;
      cost_t cp = c_pi(v, e);
      if (cp < 0) {
        if (lookAhead(e.v)) {
          i--;
          continue;
        }
        push(v, e, excess[v], true);
        if (excess[v] == 0) return;
      } else
        delta = min(delta, cp);
    }
    relabel(v, delta);
    stk.push(v);
  };
  zero_stl(pi, N);
  zero_stl(excess, N);
  for (int i = 0; i < N; i++)
    for (auto &e : g[i]) minCost += e.c * e.d, e.d *= MAX_N +
        1, eps = max(eps, e.d);
  maxFlow = getMaxFlow(N, S, T);
```

```
    while (eps > 1) {
      eps /= SCALE;
      if (eps < 1) eps = 1;
      stk = stack<int, vector<int>>();
      for (int v = 0; v < N; v++)
        for (auto &e : g[v])
          if (c_pi(v, e) < 0 && e.c > 0) push(v, e, e.c, false);
      for (int v = 0; v < N; v++)
        if (excess[v] > 0) stk.push(v);
      while (stk.size()) {
        int top = stk.top();
        stk.pop();
        discharge(top);
      }
    }
    for (int v = 0; v < N; v++)
      for (auto &e : g[v]) e.d /= MAX_N + 1, minCost -= e.c * e.d;
    minCost = minCost / 2 + negativeSelfLoop;
    return {maxFlow, minCost};
  }
};

void solve() {
  CostScalingMCMF<102, int, int, 100, 100> flow;
  int n, m;
  cin >> n >> m;
  int start = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      int inp;
      cin >> inp;
      if (inp) {
        flow.addEdge(i + 1, n + 1 + j, 1, 0);
        start++;
      } else
        flow.addEdge(i + 1, n + 1 + j, 1, 1);
    }
  }
  int counta = 0, countb = 0;
  for (int i = 0; i < n; i++) {
    int inp;
    cin >> inp;
    counta += inp;
    flow.addEdge(0, i + 1, inp, 0);
  }
  for (int i = 0; i < m; i++) {
    int inp;
    cin >> inp;
    countb += inp;
    flow.addEdge(n + i + 1, n + m + 1, inp, 0);
  }
  if (counta != countb) {
    cout << -1 << endl;
    return;
  }
  pii t = flow.maxflow(102, 0, n + m + 1);
  if (t.first != counta) {
    cout << -1 << endl;
    return;
  }
  cout << t.second + start + t.second - counta << endl;
}
```

4.2 Dinitz

```
template <class T = int>
class Dinic {
public:
  struct Edge {
```

```
    Edge(int a, T b) {
      to = a;
      cap = b;
    }
    int to;
    T cap;
  };

  Dinic(int n) {
    edges.resize(n);
    this->n = n;
  }

  T maxFlow(int src, int sink) {
    T ans = 0;
    while (bfs(src, sink)) {
      T flow;
      pt = std::vector<int>(n, 0);
      while ((flow = dfs(src, sink))) {
        ans += flow;
      }
    }
    return ans;
  }

  void addEdge(int from, int to, T cap = 1) {
    edges[from].push_back(list.size());
    list.push_back(Edge(to, cap));
    edges[to].push_back(list.size());
    list.push_back(Edge(from, 0));
  }

private:
  int n;
  std::vector<std::vector<int>> edges;
  std::vector<Edge> list;
  std::vector<int> h, pt;

  T dfs(int on, int sink, T flow = 1e9) {
    if (flow == 0) {
      return 0;
    }
    if (on == sink) {
      return flow;
    }
    for (; pt[on] < sz(edges[on]); pt[on]++) {
      int cur = edges[on][pt[on]];
      if (h[on] + 1 != h[list[cur].to]) {
        continue;
      }
      T got = dfs(list[cur].to, sink, std::min(flow,
          list[cur].cap));
      if (got) {
        list[cur].cap -= got;
        list[cur ^ 1].cap += got;
        return got;
      }
    }
    return 0;
  }

  bool bfs(int src, int sink) {
    h = std::vector<int>(n, n);
    h[src] = 0;
    std::queue<int> q;
    q.push(src);
    while (!q.empty()) {
      int on = q.front();
      q.pop();
      for (auto a : edges[on]) {
```

```
      if (list[a].cap == 0) {
        continue;
      }
      int to = list[a].to;
      if (h[to] > h[on] + 1) {
        h[to] = h[on] + 1;
        q.push(to);
      }
    }
  }
  return h[sink] < n;
 }
};

void solve() {
 int n, m;
 cin >> n >> m;
 vi a(n);
 for (int i = 0; i < n; i++) {
  cin >> a[i];
 }
 Dinic<int> flow(n + 2);
 map<int, map<int, int>> factors;
 for (int i = 0; i < n; i++) {
  for (int j = 2; j * j <= a[i]; j++) {
   while (a[i] % j == 0) {
    factors[j][i + 1]++;
    a[i] /= j;
   }
  }
  if (a[i] > 1) {
   factors[a[i]][i + 1]++;
  }
 }
 for (int i = 0; i < m; i++) {
  int u, v;
  cin >> u >> v;
  if (u % 2 == 0) {
   swap(u, v);
  }
  flow.addEdge(u, v, 100);
 }
 int ans = 0;
 for (auto t : factors) {
  Dinic<int> tempflow = flow;
  for (auto t1 : t.second) {
   if (t1.first % 2 == 0) {
    tempflow.addEdge(t1.first, n + 1, t1.second);
   } else {
    tempflow.addEdge(0, t1.first, t1.second);
   }
  }
  ans += tempflow.maxFlow(0, n + 1);
 }
 cout << ans << endl;
}
```

4.3 SCC

```
struct SCC {
    int n;
    vvi &adjLists, transposeLists;
    vi scc, leader;
    int sccCount = 0;
    vi sccSize;

    SCC(vvi& _adjLists) : n(sz(_adjLists)), adjLists(_adjLists),
        transposeLists(n), scc(n, -1), leader(n, -1) {
        for (int u = 0; u < n; u++) {
```

```
    for (int v : adjLists[u])
        transposeLists[v].push_back(u);
}

vb visited(n);
stack<int> topoSort;
function<void(int)> topoDFS = [&](int from) {
    visited[from] = true;
    for (auto to : adjLists[from]) {
        if (visited[to]) continue;
        topoDFS(to);
    }

    topoSort.push(from);
};

for (int i = 0; i < n; i++)
    if (not visited[i]) topoDFS(i);
visited.assign(n, false);
int sccPtr = 0;
sccSize.assign(n, 0);
function<void(int)> sccDFS = [&](int from) {
    scc[from] = sccPtr;
    sccSize[sccPtr]++;
    visited[from] = true;
    for (auto to : transposeLists[from]) {
        if (visited[to]) continue;
        sccDFS(to);
    }
};

while (not empty(topoSort)) {
    int i = topoSort.top();
    topoSort.pop();
    if (visited[i]) continue;
        sccDFS(i);
        leader[sccPtr] = i;
        sccPtr++;
    }

    sccCount = sccPtr;
}

int size(int index) { // Returns size of scc of index
    return sccSize[scc[index]];
}

const int& operator[](int index) {
    return scc[index];
}

vi indexInCycle;
void sccEnumeration() {
    indexInCycle.assign(n, 0);
    vb visited(n);
    int index = 0;
    function<void(int, int)> sccDFS = [&](int from, int sc) {
        indexInCycle[from] = index++;
        visited[from] = true;
        for (auto to : adjLists[from]) {
            if (scc[to] != sc) continue;
            if (visited[to]) continue;
            sccDFS(to, sc);
        }
    };

    for (int i = 0; i < sccCount; i++) {
        index = 0;
        sccDFS(leader[i], i);
    }
```

```
    }
};
```

4.4 SPFA

```
// const int INF = LONG_LONG_MAX;
vector<vector<pair<int, int>>> adj;
bool spfa(int s, vector<int>& d) {
    int n = adj.size();
    d.assign(n, INF);
    vector<int> cnt(n, 0);
    vector<bool> inqueue(n, false);
    queue<int> q;
    d[s] = 0;
    q.push(s);
    inqueue[s] = true;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        inqueue[v] = false;
        for (auto edge : adj[v]) {
            int to = edge.first;
            int len = edge.second;
            if (d[v] + len < d[to]) {
                d[to] = d[v] + len;
                if (!inqueue[to]) {
                    q.push(to);
                    inqueue[to] = true;
                    cnt[to]++;
                    if (cnt[to] > n)
                        return false; // negative cycle
                }
            }
        }
    }
    return true;
}
```

5 math

5.1 Modular

```
const int M = 1e9 + 7;
// const int M = 998'244'353
int add(int x, int y, int m = M) {
    int ret = (x + y) % m;
    if (ret < 0) ret += m;
    return ret;
}

int mult(int x, int y, int m = M) {
    int ret = (x * y) % m;
    if (ret < 0) ret += m;
    return ret;
}

int pw(int a, int b, int m = M) {
    int ret = 1;
    int p = a;
    while (b) {
        if (b & 1) ret = mult(ret, p, m);
        b >>= 1;
        p = mult(p, p, m);
    }
    return ret;
}
```

5.2 gcdextended

```
int euclid(int a, int b, int &x, int &y) {
	if (!b) return x = 1, y = 0, a;
	int d = euclid(b, a % b, y, x);
	return y -= a/b * x, d;
}
```

5.3 ntt

```
const int M = 998244353;
const int root = 3;
// (119 << 23) + 1, root = 3; // for M = 998244353
//  Can be used for convolutions modulo specific nice primes
//  of the form 2^a b+1$, where the convolution result has size
    at most 2^a$.
//  For other primes/integers, use two different primes and
    combine with CRT.
// (125000001 << 3) + 1 = 1e9 + 7, there for do not use this for
    M = 1e9 + 7
// For p < 2^30 there is also e.g. (5 << 25, 3), (7 << 26, 3),
// (479 << 21, 3) and (483 << 21, 5). The last two are > 10^9.

int add(int x, int y, int m = M) {
    int ret = (x + y) % m;
    if (ret < 0) ret += m;
    return ret;
}

int mult(int x, int y, int m = M) {
    int ret = (x * y) % m;
    if (ret < 0) ret += m;
    return ret;
}

int pw(int a, int b, int m = M) {
    int ret = 1;
    int p = a;
    while (b) {
        if (b & 1) ret = mult(ret, p, m);
        b >>= 1;
        p = mult(p, p, m);
    }
    return ret;
}

void ntt(int* x, int* temp, int* roots, int N, int skip) {
    if (N == 1) return;
    int n2 = N / 2;
    ntt(x, temp, roots, n2, skip * 2);
    ntt(x + skip, temp, roots, n2, skip * 2);
    for (int i = 0; i < N; i++) temp[i] = x[i * skip];
    for (int i = 0; i < n2; i++) {
        int s = temp[2 * i], t = temp[2 * i + 1] * roots[skip *
            i];
        x[skip * i] = (s + t) % M;
        x[skip * (i + n2)] = (s - t) % M;
    }
}

void ntt(vi& x, bool inv = false) {
    int e = pw(root, (M - 1) / sz(x));
    if (inv) e = pw(e, M - 2);
    vi roots(sz(x), 1), temp = roots;
    for (int i = 1; i < sz(x); i++) roots[i] = roots[i - 1] * e
        % M;
    ntt(&x[0], &temp[0], &roots[0], sz(x), 1);
}
```

```
// Usage: just pass the two coefficients list to get a*b (modulo
    M)
vi conv(vi a, vi b) {
    int s = sz(a) + sz(b) - 1;
    if (s <= 0) return {};
    int L = s > 1 ? 32 - __builtin_clzll(s - 1) : 0, n = 1 << L;
    if (s <= 200) { // (factor 10 optimization for |a|,|b| = 10)
        vi c(s);
        for (int i = 0; i < sz(a); i++)
            for (int j = 0; j < sz(b); j++)
                c[i + j] = (c[i + j] + a[i] * b[j]) % M;
        return c;
    }
    a.resize(n);
    ntt(a);
    b.resize(n);
    ntt(b);
    vi c(n);
    int d = pw(n, M - 2);
    for (int i = 0; i < n; i++) c[i] = a[i] * b[i] % M * d % M;
    ntt(c, true);
    c.resize(s);
    return c;
}
```

6 strings

6.1 BinaryTrie

```
struct trieobject {
  trieobject() {
    children[0] = NULL;
    children[1] = NULL;
    numelems = 0;
  };

  struct trieobject* children[2];
  int numelems;
};

struct trie {
  trieobject base;

  trie() {
    trieobject base;
  }

  void add(int x) {
    int pow2 = (1ll << 31ll);
    trieobject* temp = &base;
    while (pow2 > 0) {
      if (temp->children[1 && (x & pow2)] == NULL) {
        temp->children[1 && (x & pow2)] = new trieobject;
      }
      temp->children[1 && (x & pow2)]->numelems++;
      temp = temp->children[1 && (x & pow2)];
      pow2 /= 2;
    }
  }
  // ADD FUNCTION BELOW
};
```

7 template

```
#ifdef LOCAL
```

```
#include "include/include.h"
#else
#include <ext/pb_ds/assoc_container.hpp>
#include <bits/stdc++.h>
#endif
using namespace std;
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> o_set;
// order_of_key (val): returns the no. of values less than val
// find_by_order (k): returns the kth largest element.(0-based)
template<typename T> using minHeap = priority_queue<T,
    vector<T>, greater<T>>;
template<typename T> using maxHeap = priority_queue<T>;
#define int long long
#define all(s) s.begin(), s.end()
#define sz(s) (int) s.size()
#define testcases \
    cin >> tt;   \
    for (i = 1; i <= tt; i++)
#define fast                    \
    ios_base::sync_with_stdio(0); \
    cin.tie(NULL);              \
    cout.tie(NULL)
#define deb(a) cerr << #a << " = " << (a) << endl;
#define deb1(a)                                                  \
    cerr << #a << " = [ ";                                       \
    for (auto it = a.begin(); it != a.end(); it++) cerr << *it
        << " "; \
    cerr << "]" << endl;
typedef vector<int> vi;
typedef vector<vector<int>> vvi;
typedef pair<int, int> pii;
typedef vector<pair<int, int>> vpii;
typedef vector<bool> vb;
```

```
const int INF = LONG_LONG_MAX;
const int M = 1e9 + 7;

void solve(int tt) {

}

int32_t main() {
    fast;
    int tt = 1;
    int i = 1;
    testcases
        solve(i);
}
```

8 tree

8.1 KthAncestor

```
struct LCA {
  int n;
  vvi& adjLists;
  int lg;
  vvi up;
  vi depth;
  LCA(vvi& _adjLists, int root = 0) : n(sz(_adjLists)),
      adjLists(_adjLists) {
    lg = 1;
    int pw = 1;
    while (pw <= n) pw <<= 1, lg++;
    // lg = 20
    up = vvi(n, vi(lg));
    depth.assign(n, -1);
```

```
  function<void(int, int)> parentDFS = [&](int from, int
      parent) {
    depth[from] = depth[parent] + 1;
    up[from][0] = parent;
    for (auto to : adjLists[from]) {
      if (to == parent) continue;
      parentDFS(to, from);
    }
  };

  parentDFS(root, root);
  for (int j = 1; j < lg; j++) {
    for (int i = 0; i < n; i++) {
      up[i][j] = up[up[i][j - 1]][j - 1];
    }
  }
}

int kthAnc(int v, int k) {
  int ret = v;
  int pw = 0;
  while (k) {
    if (k & 1) ret = up[ret][pw];
    k >>= 1;
    pw++;
  }

  return ret;
}

int lca(int u, int v) {
  if (depth[u] > depth[v]) swap(u, v);
  v = kthAnc(v, depth[v] - depth[u]);
  if (u == v) return v;
  while (up[u][0] != up[v][0]) {
    int i = 0;
    for (; i < lg - 1; i++) {
      if (up[u][i + 1] == up[v][i + 1]) break;
    }

    u = up[u][i];
    v = up[v][i];
  }

  return up[u][0];
};

int dist(int u, int v) {
  return depth[u] + depth[v] - 2 * depth[lca(u, v)];
}
};
```

8.2 hld

```
struct HLD {
  template <typename T, typename F, bool VAL_ON_EDGES>
  int n, timer = 0;
  vi size, top, tin, dep, par;
  Segtree<T, F> st;
  HLD(vvi &adj, const vector<T> &arr, T id, F _m) : n(sz(adj)),
      size(n, 1), top(n), tin(n), dep(n), par(n, -1) {
    function<void(int)> dfs_hvy = [&](int v) {
      if (adj[v][0] == par[v]) swap(adj[v].front(),
          adj[v].back());
      for (auto &to : adj[v])
        if (to != par[v]) {
          par[to] = v, dep[to] = dep[v] + 1;
          dfs_hvy(to);
          size[v] += size[to];
```

```
      if (size[to] > size[adj[v][0]]) swap(to, adj[v][0]);
    }
  };
  dfs_hvy(0);
  vector<T> temp;
  temp.reserve(n);
  function<void(int)> dfs_hld = [&](int v) {
    tin[v] = timer++;
    temp.pb(arr[v]);
    for (auto &to : adj[v])
      if (to != par[v]) {
        top[to] = (to == adj[v][0] ? top[v] : to);
        dfs_hld(to);
      }
  };
  dfs_hld(0);
  st = Segtree<T, decltype(_m)>(temp, id, _m);
}
template <class B>
int process(int u, int v, B op) {
  for (; top[u] != top[v]; v = par[top[v]]) {
    if (dep[top[u]] > dep[top[v]]) swap(u, v);
    op(tin[top[v]], tin[v]);
  }
  if (dep[u] > dep[v]) swap(u, v);
  if (!VAL_ON_EDGES or u != v) op(tin[u] + VAL_ON_EDGES,
      tin[v]);
  return u;
}
int lca(int u, int v) {
  return process(u, v, [&](int, int) {});
}
T query(int u, int v) {
  T ans = st.identity;
  process(u, v, [&](int l, int r) {
    ans = st.merge(ans, st.query(l, r));
  });
  return ans;
}
void update(int v, T val) {
  st.update(tin[v], val);
}
};
```