RAUCODER 2025

EDITORIAL

24 mai 2025

Problema 1. Bomboane

Autor: Crișan Daniela Alexandra, conf. dr., Universitatea Româno-Americană

Cuvinte cheie: Principiul includerii și excluderii, Stars and Bars, Măști de biți, Aritmetica modulară, Exponențiere rapidă

Indicații

Pasul 1. Determinarea numărului de elevi

Pentru a afla numărul de elevi, trebuie să determinăm câte numere între 1 și N sunt coprime cu Q. Pentru aceasta vom folosi Principiul Includerii și Excluderii.

Vom afla câte numere între 1 și N nu au niciun divizor comun cu Q, în afară de 1.
Dacă $Q=p_1^{a1}\cdot p_2^{a2}\cdot\ldots\cdot p_q^{aq}$, atunci vom scădea din N toate numerele care **NU** sunt coprime cu Q, adică toate cele care sunt divizibile cu cel puțin unul din factorii primi ai lui Q: p_1, p_2, ..., p_q.

$$nr\_elevi = N - \sum|Ai| + \sum|Ai \cap Aj| - \sum|Ai \cap Aj \cap Aq| + \ldots$$

unde Ai este mulțimea numerelor între 1 și N divizibile cu p_i.

Vom calcula acest număr astfel: considerăm toate submulțimile de factori primi $S \subseteq \{p_1,p_2,\ldots,p_q\}$ și calculăm:

- d = produsul tuturor primilor din S
- partea întreagă [N/d] va fi numărul de multipli ai lui d în intervalul [1,N]
- adunăm sau scădem [N/d] în funcție de paritatea mărimii (cardinalului) lui S (un număr prim → scădem, două numere prime → adunăm etc.)

Pentru a genera toate submulțimile S se vor folosi măști de biți.

Pasul 2. Verificarea dacă bomboanele ajung

Numărul de colegi este k = nr_elevi -1, așadar bomboanele ajung dacă numărul lor, M, este cel puțin egal cu k.

Pasul 3. Împărțirea bomboanelor – Număr de moduri

Dacă bomboanele sunt suficiente, trebuie să aflăm în câte moduri putem împărți M bomboane către k = nr_colegi, cu cel puțin una pentru fiecare. Avem o problemă clasică de numărare a partițiilor cu restricții:
C(M−1, k-1) pentru care putem apela la metoda Stars and Bars.

Pentru a calcula eficient binomul modulo MOD = 10^9+7 se vor folosi aritmetica modulară și exponențierea rapidă.

Complexitate timp: $O(\sqrt{Q} + 2^{\wedge}q + M + log\ MOD)$, unde q = numărul de factori primi ai lui Q
Complexitate memorie: O(q)
(numerele Q ≤ 10^6 au cel mult 7 factori primi)

Soluție

```
#include <fstream>
using namespace std;

ifstream in("bomboane.in");
ofstream out("bomboane.out");

#define ll long long
const int MOD = 1e9 + 7;

static int count_coprime_with_Q(int N, int Q) {
    int primes[10], cnt=0;
    int q = Q;
    for (int p = 2; p * p <= q; ++p) {
        if (q % p == 0) {
            primes[cnt++] = p;
            while (q % p == 0)
                q /= p;
        }
    }
    if (q > 1)
        primes[cnt++] = q;

    int total = 0;
    int subsets = 1 << cnt;
    for (int mask = 1; mask < subsets; ++mask) {
        int mult = 1, bits = 0;
        for (int i = 0; i < cnt; ++i) {
            if (mask & (1 << i)) {
                mult *= primes[i];
                bits++;
            }
        }
        int x = N / mult;
        if (bits % 2 == 1) total += x;
        else total -= x;
    }
    return N - total;
}

static ll power(ll base, ll exp) {
    ll result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) result = (result * base) % MOD;
        base = (base * base) % MOD;
        exp /= 2;
    }
    return result;
}

static ll comb(int n, int k) {
    if (k > n || k < 0) return 0;
    ll num = 1, den = 1;
```

```
    for (int i = 0; i < k; i++) {
        num = (num * (n - i)) % MOD;
        den = (den * (i + 1)) % MOD;
    }
    return (num * power(den, MOD - 2)) % MOD;
}

int main()
{
    int Q, N, M;
    in >> Q >> N >> M;
    int C = count_coprime_with_Q(N, Q);
    out << C << "\n";
    if (M >= C - 1)
        out << comb(M - 1, C - 2);
    else
        out << "NU";

    return 0;
}
```

Problema 2. Matrice Interesantă

Autor: Lorintz Alexandru, Universitatea Babeș-Bolyai

Cuvinte cheie: Two pointers, Range minimum query, Măști de biți

Indicații

La o primă analiză a restricțiilor problemei, se observă că numerele din matrice au valori mici, ceea ce indică către o soluție care ar putea profita de această limitare.

Pentru început, se pot fixa liniile care limitează subamtricele căutate superior, respectiv inferior, obținându-se astfel **O(N^2)** perechi. Acum trebuie găsită o abordare eficientă pentru a număra perechi de coloane care împreună cu aceste linii determina submatrice cu proprietatea din enunț. Problema aici este că proprietatea din enunț este mult prea restrictivă, dar poate fi reformulată, calculând submatrice cu cel mult **X** elemente distincte, rezultatul căutat devenind astfel diferența între rezultatele noii probleme pentru **X=K** și **X=K-1**.

Pentru noua problemă definită, se poate folosi **tehnica celor doi pointeri** în continuare, dar este nevoie de o modalitate rapidă de a afla la fiecare pas câte elemente distincte se află într-o submatrice fixată, interogare necesară pentru modalitatea de actualizare a pointerilor. Aici intervine limitarea valorilor din matrice. Practic, pentru o submatrice se poate folosi o **mască de biți** care are un bit activ dacă numărul valoarea acelui bit apare în submatrice. Pentru a calcula această mască rapid pentru o submatrice arbitrară, se poate precalcula o structură de date similară cu un **Range minimum query bidimensional**. De asemenea, este de menționat că ne preocupă numărul de biți setați ai unei măști arbitrare, dar acesta poate fi ușor aflat folosind funcția predefinită **__builtin_popcountll**, care este foarte rapidă în practică. Astfel, complexitatea totală este **O(N^2 * log^2(N) + N^2 * M)** și **O(N^2 * log^2(N))** spațiu de la calcularea structurii de **Range minimum query**.

Soluție

```
#include <fstream>
#include <cstdint>
using namespace std;

ifstream fin("matriceinteresanta.in");
ofstream fout("matriceinteresanta.out");

const int kN = 300;
const int kLog = 8;
int n, m, a[1 + kN][1 + kN];
int lg2[1 + kN];
int64_t rmq[1 + kLog][1 + kLog][1 + kN][1 + kN];

int64_t query(int x1, int y1, int x2, int y2) {
    if (x2 < x1 || y2 < y1) {
        return 0;
    }

  int h1 = lg2[x2 - x1], h2 = lg2[y2 - y1];
  return rmq[h1][h2][x1][y1] | rmq[h1][h2][x2 - (1 << h1) + 1][y1] |
           rmq[h1][h2][x1][y2 - (1 << h2) + 1] | rmq[h1][h2][x2 - (1 << h1) + 1][y2 - (1
<< h2) + 1];
}

void precalc() {
```

```
  for (int i = 2; i <= kN; ++i) {
    lg2[i] = lg2[i >> 1] + 1;
  }

  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) {
      rmq[0][0][i][j] = 1LL << a[i][j];
    }
  }

  for (int h1 = 0; h1 <= kLog; ++h1) {
    for (int h2 = 0; h2 <= kLog; ++h2) {
      if (h1 > 0 || h2 > 0) {
        for (int i = 1; i <= n - (1 << h1) + 1; ++i) {
          for (int j = 1; j <= m - (1 << h2) + 1; ++j) {
            rmq[h1][h2][i][j] = query(i, j, i + (1 << h1) - 1, j + (1 << h2) - 1);
          }
        }
      }
    }
  }
}

int64_t solveAtMostK(int k) {
  int64_t res = 0;

  for (int l1 = 1; l1 <= n; ++l1) {
    for (int l2 = l1; l2 <= n; ++l2) {
        int left = 1;

        for (int right = 1; right <= m; ++right) {
            while (left <= right && __builtin_popcountll(query(l1, left, l2, right)) > k)
{
                left += 1;
            }

            res += right - left + 1;
        }
    }
  }

  return res;
}

int main() {
  int k;
  fin >> n >> m >> k;

  for (int i = 1; i <= n; ++i) {
    for (int j = 1; j <= m; ++j) {
        fin >> a[i][j];
    }
  }

  precalc();

  fout << solveAtMostK(k) - solveAtMostK(k - 1) << '\n';

  return 0;}
```

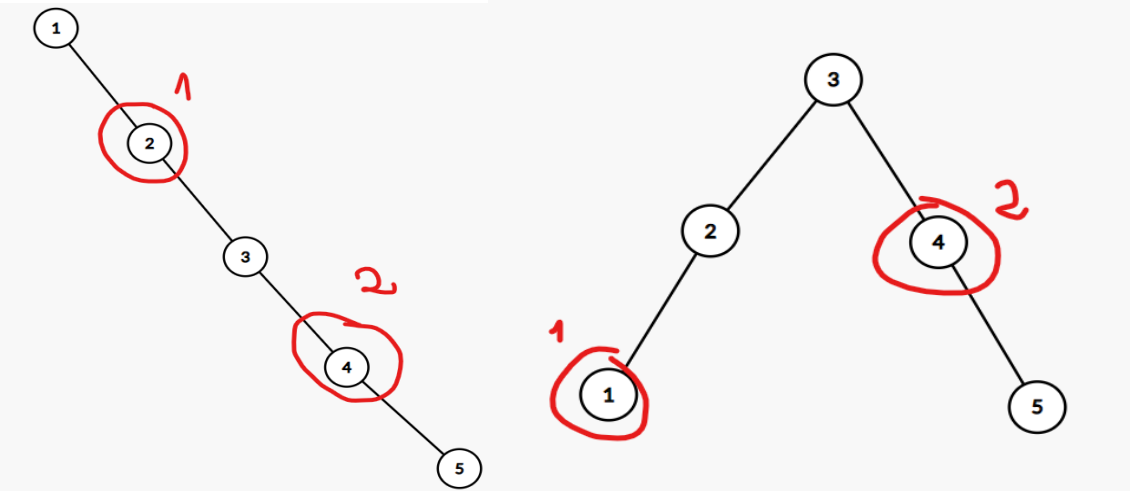
Problema 3. Allca

Autor: Carole Șerban, Delft University of Technology

Cuvinte cheie: Lowest common ancestor (Lca), Divide and Conquer, Small-to-large

Soluție completă:

Dacă pornim de la o mulțime nevidă de noduri și adăugăm un nod, se pot întâmpla două lucruri - lca-ul va rămâne același sau va deveni strămoșul lca-ului precedent.
Să pornim de la ipoteza că avem 2 noduri. Sunt 2 situații posibile:
1. Unul dintre noduri se află pe drumul de la rădăcină la celălalt.
2. Niciunul dintre noduri nu e strămoșul celuilalt.



Observăm că în ambele cazuri, urcarea nodului 2 schimbă lca-ul doar atunci când acesta devine unul din strămoșii primului lca dintre cele două noduri. De asemenea, odată schimbat, va fi mereu egal cu nodul 2. Această observație ne duce la ideea aplicării metodei divide and conquer pentru numărare. Putem afla lca-ul dintre două noduri în $O(1)$ precalculând range minimum query pe euler tour.

Ideea cu divide and conquer este să calculăm pentru fiecare call, suma lca-urilor subsecvențelor care au capătul din stânga în [st,mij], iar pe cel din dreapta în [mij+1,dr], celelalte fiind calculate deja în alt call de divide. Facem apeluri succesive în care împărțim intervalele în [st,mij] și [mij+1,dr] și iterăm după indicii i din [st,mij]. Pentru fiecare, căutăm binar primul j > mij cu proprietatea că lca-ul nodurilor de la i la j este egal cu lca-ul nodurilor de la mij+1 la j. În cazul oricărui k >= j, știm din proprietatea menționată în paragraful precedent că lca-ul de la i la k este egal cu cel de la mij + 1 la k , deci precalculăm pentru fiecare element > mij sp[x] = suma după lca de la mij + 1 la y, pentru oricare y >= x, simplu prin sume parțiale. Pentru elementele mai mici ca j (cel pe care l-am căutat binar) știm că lca-ul de la i la ele este egal pentru toate. Și deci ajungem la relația pentru elementul i per call de divide ca fiind:
Suma după subsecvențele care încep în i = sp[j] + (j-1-mij) * (lca-ul de la i la mij+1).
Cum trebuie să căutăm binar pentru fiecare element în log call-uri de divide ajungem la complexitatea $O(M \cdot \log^2(M) + N \cdot \log(N))$. Această soluție nu obține punctaj maxim.
Optimizarea necesară pentru 100 de puncte este să realizăm că dacă putem urca și nodul 1, acel j căutat binar poate doar să se mute la dreapta, deoarece 2 va ajunge ori la fel de repede să depășească lca-ul, ori mai târziu.

Așa că iterăm cu i de la mij la st, pentru i = mij căutăm j brut și după doar îl mutăm la dreapta când e cazul. Complexitatea finală este $O(M \log M + N \log N)$.

Soluție alternativă (Prosie Radu-Teodor, FMI Unibuc):

Ne dorim să calculăm pentru fiecare nod contribuția sa la răspuns.
Dacă notăm cu S_i = numărul de subsecvențe din A pentru care i este strămoș atunci contribuția lui i poate fi obținută după formula : $S_i - \sum_{F\ fiu\ al\ lui\ i} S_F$. Pentru a calcula S_i putem folosi DSU cu tehnica small-to-large :

Facem DFS și ținem pentru fiecare nod un DSU care reprezintă subsecvențele compacte create de pozițiile nodurilor din subarborele lui în cadrul șirului A. De asemenea, trebuie să menținem și numărul de subsecvențe ale acestor subsecvențe compacte, lucru ce poate fi recalculat ușor la fiecare unire. Dupa ce am facut DFS în fiul lui i, unim DSU-ul lui i și al fiului lui i inserându-l pe cel cu mai puține elemente în cel cu mai multe (eventual interschimbând, dacă fiul are DSU-ul mai mare).

A se nota că trebuie să menținem DSU-urile puțin diferit pentru a avea memorie liniară în numărul de noduri care ne interesează din subarbore, lucru pentru care putem folosi hashing. Este probabil ca soluția să nu intre dacă folosiți hashmap-ul din STL, dar intră dacă îl scrieți de mână sau folosiți gp_hash_table din __gnu_pbds.

Complexitate : $O(M \log M \cdot \alpha(M) + N)$

Soluții parțiale:

Pentru subtask-ul 1, observăm că în cazul oricărei subsecvențe, răspunsul este capătul intervalului, deci putem să calculăm contribuția pentru capete (sumă după A[i] * i).

Subtask-ul 2 poate fi rezolvat în $O(N + M)$, cu o stivă - calculăm vectorii st[i] și dr[i] = cel mai apropiat element la stânga/dreapta de i care are un nivel mai mic (la stânga) și mai mic sau egal (la dreapta). Apoi contribuția per element este egală cu (dr[i] - i) * (i – st[i]). Orice soluție cu un logaritm în plus ar trebui să ia punctele totuși.

Pentru subtask-urile 3 și 4, orice soluție pătratică, sau $O(M^2 \cdot \log(M))$ este îndeajuns de bună. O soluție simplă și scurtă este doar să pornim de la un nod de start (cel mai din stânga al subsecvenței), să îl punem ca lca, și odată cu adăugarea unui nod să urcăm lca-ul până când este și strămoșul noului nod. Odată ce este, oprim urcarea și îl adăugăm la sumă. Soluția aceasta este $O\big(M \cdot (N + M)\big)$, deoarece urcarea pentru fiecare nod de start se amortizează la $O(H)$, unde H = distanța de la rădăcină la acest nod de inceput.

Soluție Divide and Conquer

```
#include <fstream>
#include <vector>
using namespace std;
ifstream cin("allca.in");
ofstream cout("allca.out");
const int nmax = 2e5 + 1;
int n , m , a , b , ind;
int v[2*nmax] , d[nmax] , p[2*nmax] , tin[nmax] , tout[nmax];
int rmq[19][2*nmax], lg[2*nmax] , l;
long long ans , sp[2*nmax];
vector <int> g[nmax];

void dfs(int x , int p)
{
    rmq[0][++ind] = x;
    tin[x] = ind;
    d[x] = d[p] + 1;
    for(auto it : g[x])
        if(it != p)
        {
            dfs(it,x);
            rmq[0][++ind] = x;
        }
    tout[x] = ind;
}

int helper(int x , int y)
{
    if(d[x] < d[y]) return x;
    else return y;
}

int lca(int x , int y)
{
    if(tin[x] > tin[y])
    {
        l = lg[tin[x]-tin[y]+1];
        return helper(rmq[l][tin[x]],rmq[l][tin[y]+(1<<l)-1]);
    }
    else
    {
        l = lg[tin[y]-tin[x]+1];
        return helper(rmq[l][tin[y]],rmq[l][tin[x]+(1<<l)-1]);
    }
}

void divide(int st , int dr)
{
    if(st == dr)
    {
        ans += v[st];
    }
    else
    {
        int mij = (st+dr)/2;
        int nod = v[mij+1];
        for(int i = mij+1 ; i <= dr ; ++i)
```

```
        {
            nod = lca(nod,v[i]);
            p[i] = nod;
            sp[i] = p[i];
        }
        for(int i = dr-1 ; i > mij ; --i) sp[i] += sp[i+1];
        int j;
        for(int i = mij ; i >= st ; --i)
        {
            if(i == mij)
            {
                j = mij+1;
                nod = v[mij];
            }
            else nod = lca(nod,v[i]);
            while(j <= dr)
            {
                if(p[j] == lca(nod,p[j])) break;
                ++j;
           }
           if(j <= dr) ans += sp[j];
           ans += 1LL*(j-mij-1)*lca(nod,v[mij+1]);
        }
        divide(st,mij);
        divide(mij+1,dr);
    }
}

int main()
{
    cin >> n;
    for(int i = 1 ; i < n ; ++i)
    {
        cin >> a >> b;
        g[a].push_back(b);
        g[b].push_back(a);
    }
    cin >> m;
    for(int i = 1 ; i <= m ; ++i) cin >> v[i];
    dfs(1,0);
    lg[0] = -1;
    for(int i = 1 ; i <= 2*n-1 ; ++i) lg[i] = lg[i/2]+1;
    for(int i = 1 ; i <= lg[2*n-1] ; ++i)
    {
        for(int j = (1<<i) ; j <= 2*n-1 ; ++j)
        {
            rmq[i][j] = helper(rmq[i-1][j],rmq[i-1][j-(1<<(i-1))]);
        }
    }
    divide(1,m);
    cout << ans;
    return 0;
}
```

Soluție Disjoint Set Union

```
#pragma GCC optimize("Ofast,unroll-loops,inline")
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/hash_policy.hpp>
#include <fstream>
#include <utility>
#include <vector>

using namespace __gnu_pbds;
using namespace std;

ifstream cin("allca.in");
ofstream cout("allca.out");

struct custom_hash {
  size_t operator()(const int x) const noexcept {
    return static_cast<unsigned>(x);
  }
};

class lekee {
  gp_hash_table<int, int, custom_hash> *t_ptr;
  gp_hash_table<int, int, custom_hash> *sz_ptr;

  int root_impl(const int x) noexcept {
    int &parent_ref = (*t_ptr)[x];
    if (parent_ref == 0) return x;
    return parent_ref = root_impl(parent_ref);
  }
  void unite(int a, int b) noexcept {
    a = root_impl(a);
    b = root_impl(b);
    if (a == b)
      return;

    const int sa = (*sz_ptr)[a];
    const int sb = (*sz_ptr)[b];

    coef -= 1ll * sa * (sa + 1);
    coef -= 1ll * sb * (sb + 1);

    if (sb > sa)
      std::swap(a, b);
    (*sz_ptr)[a] += (*sz_ptr)[b];
    (*t_ptr)[b] = a;

    coef += 1ll * (*sz_ptr)[a] * ((*sz_ptr)[a] + 1);
  }

public:
  long long coef;

  lekee() : t_ptr(nullptr), sz_ptr(nullptr), coef(0) {}
  ~lekee() {
    delete t_ptr;
    delete sz_ptr;
  }
```

```
  lekee(lekee &&other) noexcept : t_ptr(nullptr), sz_ptr(nullptr), coef(0) {
    std::swap(t_ptr, other.t_ptr);
    std::swap(sz_ptr, other.sz_ptr);
    std::swap(coef, other.coef);
  }

  lekee &operator=(lekee &&other) noexcept {
    if (this != &other) {
      delete t_ptr;
      delete sz_ptr;
      t_ptr = nullptr;
      sz_ptr = nullptr;
      coef = 0;
      std::swap(t_ptr, other.t_ptr);
      std::swap(sz_ptr, other.sz_ptr);
      std::swap(coef, other.coef);
    }
    return *this;
  }

  lekee(const lekee &) = delete;
  lekee &operator=(const lekee &) = delete;

  void activate(const int x) {
    if (!sz_ptr)
      sz_ptr = new gp_hash_table<int, int, custom_hash>();
    if (!t_ptr)
      t_ptr = new gp_hash_table<int, int, custom_hash>();
    (*sz_ptr)[x] = 1;
    coef += 2;

    if (sz_ptr->find(x - 1) != sz_ptr->end())
      unite(x - 1, x);
    if (sz_ptr->find(x + 1) != sz_ptr->end())
      unite(x + 1, x);
  }

  [[nodiscard]] long long get_coef() const noexcept { return coef; }

  void swap(lekee &other_leke) noexcept {
    std::swap(t_ptr, other_leke.t_ptr);
    std::swap(sz_ptr, other_leke.sz_ptr);
    std::swap(coef, other_leke.coef);
  }

  void clear_content() noexcept {
    delete t_ptr;
    t_ptr = nullptr;
    delete sz_ptr;
    sz_ptr = nullptr;
    coef = 0;
  }
};

vector<vector<int>> g_storage, poz_storage, sub_storage;
vector<lekee> susu_storage;
vector<long long> ans_storage;

void meld(const int node_idx_a, const int node_idx_b) {
  if (sub_storage[node_idx_a].size() < sub_storage[node_idx_b].size()) {
```

```
    sub_storage[node_idx_a].swap(sub_storage[node_idx_b]);
    susu_storage[node_idx_a].swap(susu_storage[node_idx_b]);
  }
  for (auto &it : sub_storage[node_idx_b]) {
    susu_storage[node_idx_a].activate(it);
    sub_storage[node_idx_a].push_back(it);
  }
}

void dfs(const int x_node, const int p_node = 0) {
  for (int &val_idx : poz_storage[x_node]) {
    susu_storage[x_node].activate(val_idx);
    sub_storage[x_node].push_back(val_idx);
  }
  for (int child_node : g_storage[x_node]) {
    if (child_node == p_node) continue;
    dfs(child_node, x_node);
    meld(x_node, child_node);
    susu_storage[child_node].clear_content();
    sub_storage[child_node].clear();
    sub_storage[child_node].shrink_to_fit();
 }
  ans_storage[x_node] += susu_storage[x_node].get_coef();
  if (p_node != 0) {
    ans_storage[p_node] -= susu_storage[x_node].get_coef();
  }
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr);

  int n_nodes, m_elements;
  cin >> n_nodes;

  g_storage.resize(n_nodes + 1);
  poz_storage.resize(n_nodes + 1);
  sub_storage.resize(n_nodes + 1);
  susu_storage.resize(n_nodes + 1);
  ans_storage.resize(n_nodes + 1, 0);

  for (int i = 1; i < n_nodes; i++) {
    int u, v;
    cin >> u >> v;
    g_storage[u].push_back(v);
    g_storage[v].push_back(u);
  }
  cin >> m_elements;
  for (int i = 1; i <= m_elements; i++) {
    int node_val_appears_at;
    cin >> node_val_appears_at;
    poz_storage[node_val_appears_at].push_back(i);
  }
  dfs(1);
  long long total = 0;
  for (int i = 1; i <= n_nodes; i++)
    total += ans_storage[i] * i;
  cout << total / 2 << "\n";
  return 0;}
```